FLUJO DE DISEÑO PARA EL DESARROLLO DE APLICACIONES EN TIEMPO REAL USANDO LENGUAJE ADA

FELIPE ANDRES VIVAS CAICEDO

UNIVERSIDAD AUTÓNOMA DE OCCIDENTE
FACULTAD DE INGENIERÍA
UNIDAD ACADEMICA DE INGENIERÍA ELECTRICA Y ELECTRÓNICA
SANTIAGO DE CALI
2003

FLUJO DE DISEÑO PARA EL DESAROLLO DE APLICACIONES EN TIEMPO REAL USANDO LENGUAJE ADA

FELIPE ANDRES VIVAS CAICEDO

Trabajo de grado para optar por el titulo de ingeniero electrónico

Director
DIEGO MARTINEZ
Jefe del Área de Electrónica Digital

UNIVERSIDAD AUTÓNOMA DE OCCIDENTE
FACULTAD DE INGENIERÍA
UNIDAD ACADEMICA DE INGENIERÍA ELECTRICA Y ELECTRÓNICA
SANTIAGO DE CALI
2003

Nota de aceptación:

Trabajo aprobado por el comité de grado en cumplimiento de los requisitos exigidos por la Universidad Autónoma de Occidente para optar por el titulo de Ingeniero Electrónico .

DRAGO DUSSICH

JURADO

ALEXANDER CASTILLO

JURADO

Santiago de Cali, 20 de Noviembre del 2003

AGRADECIMIENTOS

A Mis padres y a mi hermana por su amor, sacrificio y por creer en mi en algunos momentos difíciles de mi vida.

A Mary por brindarme su amor, comprensión y alentarme en momentos de flaqueza.

A mis fieles amigos Andrés, Jorge, Norberto y Drago por sacrificar parte de su tiempo al ofrecerme su ayuda.

Y sobre todas las cosas agradezco a Dios por ser aquel faro que guía mi destino y siempre me acompaña.

CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ANEXOS

RESUMEN

En este trabajo de grado se propone un estilo formal de diseño para el desarrollo de aplicaciones para Tiempo Real usando lenguaje Ada.

Dada la complejidad del desarrollo de los sistemas analizados y debido al alto grado de exigencia temporal que los mismos requieren, en este trabajo se presenta un proceso de desarrollo de software que permite especificar todos los aspectos que involucran el diseño de este tipo de aplicaciones. Para tal efecto, se utiliza el método de diseño HRT-HOOD, el cual permite diseñar soluciones basadas en objetos para sistemas críticos de Tiempo Real.

Se plantea el uso del lenguaje Ada como un lenguaje de síntesis para la programación de aplicaciones en Tiempo Real, debido a que este lenguaje cuenta con ciertas características que lo hacen atractivo para el desarrollo de este tipo de sistemas, dichas características se basan en el cumplimiento de especificaciones para Tiempo Real, como lo son las normas Posix, las cuales entre otras cosas permiten establecer parámetros de sincronización y comunicación entre procesos para Tiempo Real. Los cuales son soportados por librerías que contiene el Gnat(Compilador del lenguaje de programación Ada) como son la libreria Gnull y la Gnatrl.

INTRODUCCIÓN

Los sistemas de Tiempo Real son sistemas complejos que deben cumplir ciertas restricciones temporales impuestas por el diseñador en la fase de diseño del mismo. Estos sistemas deben ser predecibles y estables en el momento de su implementación, para ello se debe contar con un estilo de diseño que garantice un buen desempeño en el funcionamiento de dichos sistemas. En este documento, se va a tratar un tema importante no solo para la industria colombiana sino también para todo aquel desarrollador de Sistemas para Tiempo Real que este interesado en adoptar un estilo de diseño que le permita reducir costos en cuanto tiempo de diseño e implementación, de un sistema de este tipo. Además, se mostrará en detalle la definición y las características de un Sistema de Tiempo Real, lo cual permitirá comprender la diferencia entre este tipo de sistemas y un Sistema en Línea.

Uno de los requisitos primordiales de los Sistemas en Tiempo Real es que sus componentes tengan un comportamiento predecible. Un componente básico para evaluar el comportamiento predecible de los sistemas es el análisis de la planificabilidad, si esta arroja buenos resultados, el sistema cumplirá con los plazos de todas sus tareas. Dado lo anterior se puede decir que los Sistemas en Tiempo Real aplican a todos aquellos procesos que requieran cumplir de manera relevante con plazos temporales más que con rapidez; Por ejemplo sistemas de control industrial, sistemas utilizados en la medicina moderna para monitorear el ritmo cardíaco de un paciente o para controlar plantas de energía e incluso medios de transporte como aviones y trenes de última generación.

Debido al grado de compromiso de los ejemplos anteriores para desarrollar Sistemas en Tiempo Real el diseñador debe tener a la mano un estilo de diseño que se convierta en su Bitácora de Vuelo, no solo con el objetivo de poder diseñar en plazos de tiempo razonables, sino también contar con un proceso de desarrollo que permita hacer desde una definición detallada de los requerimientos del sistema hasta una implementación óptima del mismo, sin dejar detalles al azar y cumpliendo con todos los plazos temporales y funcionales impuestos por la aplicación.

Los diseñadores necesitan herramientas para razonar, simular, analizar, verificar, validar y comprobar el comportamiento adecuado de las especificaciones realizadas. Actualmente muchos desarrollos de software usan métodos de especificaciones informales que crean ambigüedades, inconsistencias y análisis limitados (González, 1999). Este tipo de metodología usada para el desarrollo de Sistemas en Tiempo Real carece de definiciones formales y mecanismos a la hora de expresar ciertos niveles de concurrencia, sincronización y comunicación entre hilos.

Por tal motivo, los productos obtenidos no corresponden completamente a las especificaciones iniciales y generalmente presentan un número significativo de errores. El número de errores en la especificación y diseño puede ser reducido usando técnicas de verificación y análisis. Además, la utilización de los métodos formales se ha incrementado, aunque su complejidad desde el punto de vista matemático no ha permitido una gran aceptación en el ambiente industrial (González, 1999).

Lo ideal es contar con un método de diseño que aporte la facilidad de uso de los métodos informales, pero que le brinde al diseñador la capacidad de garantizar un análisis y una validación de las especificaciones realizadas, de tal forma que a la hora de la implementación del diseño, se cumpla con todas las funcionalidades descritas por las especificaciones del proyecto. Un método de estas características se convertiría en una buena opción a nivel industrial puesto que los diseños carecerían de grandes y complejos análisis matemáticos, aunque se podría contar con toda la filosofía que requieren los diseños de Tiempo Real, logrando desarrollar software de calidad en el medio industrial.

En el desarrollo de esta tesis se presentará la manera de como realizar un buen diseño de un Sistema para Tiempo Real de una forma óptima y eficiente, además se dará a conocer en detalle la utilización del Lenguaje Ada como lenguaje de programación para Sistemas en Tiempo Real, de acuerdo a ciertas características que convierten al lenguaje en una buena alternativa para desarrollar este tipo de sistemas.

1. PLANTEAMIENTO DEL PROBLEMA

Algunas de las implementaciones realizadas para dar solución a los problemas presentes en el campo de los sistemas en Tiempo Real carecen de una planeación adecuada, lo cual ha generado desarrollos que presentan problemas al momento de la implementación.

Entre las posibles causas se han detectado las siguientes:

- Un mal análisis del comportamiento temporal del proceso, lo que conlleva a un errado criterio a la hora de establecer la validación del comportamiento temporal del mismo.
- Un errado diseño arquitectónico del sistema por la carencia de un método adecuado de diseño, lo cual conlleva a una implementación poco robusta por la falta de una buena solidez en la concurrencia de la aplicación y soluciones costosas en tiempo y dinero.
- Utilización de lenguajes secuénciales tradicionales como es el caso de lenguaje C, los cuales ofrecen soluciones muy complejas a la hora de programar aplicaciones en Tiempo Real, ya que los lenguajes secuénciales requieren de líneas de código especificas que permiten a la aplicación hacer muchos llamados al sistema para establecer algún tipo de gobierno sobre el planificador del sistema operativo.

Estos detalles hacen que los diseñadores tarden mucho tiempo en encontrar la solución a un problema, puesto que puede que demoren mucho tiempo en terminar de depurar la aplicación debido a todas las variables que deben ser consideradas. Esto se debe en gran parte a que las metodologías tradicionales no permiten enfrentar en forma óptima los problemas que plantean las actuales necesidades de la sociedad y a una mala escogencia de las herramientas de especificación, validación y síntesis. Dado lo anterior, el problema a enfrentar es la falta de un método adecuado para especificar y validar sistemas de control en Tiempo Real, utilizando Ada como lenguaje de síntesis.

2. MARCO TEORICO

2.1 QUE ES UN SISTEMA DE TIEMPO REAL?

Actualmente muchas instituciones tienen en uso sistemas descritos como Sistemas en Línea (on line) y Sistemas en Tiempo Real, este tipo de terminología se presta para confusiones, pues, las personas por lo general creen que ambos términos significan lo mismo. Se puede definir un sistema en línea como aquel en que los datos de entrada pasan directamente desde su lugar de origen y se trasmiten en forma directa a la salida del sistema o al lugar de destino donde se utilizaran posteriormente. De esta manera se evitan en gran parte las etapas intermedias de almacenamiento de datos para su posterior procesamiento (Martin, 1980).

Por su parte, el término Tiempo Real se utiliza con excesiva frecuencia para un gran número de aplicaciones que tienen una respuesta rápida, se considera que los sistemas de Tiempo Real son aquellos que tienen una respuesta acotada y predeterminada. A pesar, de que una característica de éstos debe ser la rapidez y la eficiencia, éstas no definen un sistema de Tiempo Real sino existe un análisis de los tiempos de respuesta del sistema, he aquí la diferencia con los Sistemas en Línea (Martin,1980).

"Para los sistemas de Tiempo Real también conocidos como sistemas STR no sólo es importante la validez lógica de la respuesta que proporciona, sino que ésta debe generarse antes de un plazo determinado"[1] . "La obtención de una respuesta en un instante de tiempo posterior al plazo, puede ser más perjudicial que obtener una respuesta imprecisa pero

[1] STANKOVIC, J. A. Misconceptions about real-time computing,, citado por BALBASTRE, Patricia. Modelo De Tareas Para La Integración Del Control y La Planificación En Sistemas De Tiempo Real. Valencia, 2002 ,165 p. Tesis Doctoral (Ingeniero Electrónico). Universidad Politécnica de Valencia. Departamento de Informática de Sistemas y Computadores.

dentro del plazo"[2]. Es decir, un Sistema en Tiempo Real es cualquier sistema que tiene que responder a estímulos generados externamente dentro de un plazo especificado y finito.
La eficiencia de un sistema en Tiempo Real depende:

- Del *resultado lógico* de la computación.
- Del *tiempo* en que este resultado tarda en generarse.

Un Sistema de Tiempo Real es un sistema informático en el que es significativo el tiempo en el que se producen sus acciones. No es suficiente que las acciones del sistema sean correctas desde el punto de vista lógico o algorítmico, sino que, además, deben producirse dentro de un intervalo de tiempo determinado. Esto es debido a que el sistema está conectado a un proceso externo del que recibe estímulos a los que debe responder con suficiente rapidez para evitar que evolucione a un estado indeseable. Más que ser rápido, un sistema en Tiempo Real debe ser *predecible*. En función de que tan estricto es el cumplimiento de los plazos del sistema, los sistemas de Tiempo Real se clasifican en:

- Los Sistemas de Tiempo Real Duro o Crítico (Hard Real - Time Systems), en los cuales, la pérdida de un plazo puede ocasionar un fallo catastrófico e irrecuperable.
- Los Sistemas de Tiempo Real Blando o No Crítico(Soft Real – Time Systems), en estos sistemas se programan tareas con plazos de cumplimiento deseable pero no crítico, es decir si los plazos no se cumplen el sistema falla pero no de forma catastrófica.

En los sistemas de control, la pérdida de un plazo, es decir, el envío de la acción de control a destiempo, degrada el desempeño del sistema en comparación con los análisis realizados en herramientas de simulación, llegando incluso a regiones de inestabilidad(Zhang, 2001).Por otro lado los Sistemas de Tiempo Real No Critico (Soft Real-Time Systems), a diferencia de los Sistemas de Tiempo Real Critico, no representan mayor peligro si se

[2] HARMON, M. y WHALLEY, D. A retargetable technique for predicting execution time, citado por BALBASTRE, Patricia. Modelo De Tareas Para La Integración Del Control y La Planificación En Sistemas De Tiempo Real. Valencia, 2002 ,165 p. Tesis Doctoral (Ingeniero Electrónico). Universidad Politécnica de Valencia. Departamento de Informática de Sistemas y Computadores.

presenta la pérdida de un plazo, entre estos sistemas tenemos por lo general sistemas de monitoreo.

Dado lo anterior podemos decir que al analizar un sistema en Tiempo Real se debe tener especial cuidado tanto en el comportamiento funcional como en el comportamiento temporal de dicho sistema, es por esto que para el estudio, e implementación de los sistemas en Tiempo Real se debe contar no solo con una metodología de diseño que de alguna manera permita el análisis de objetos característicos de este tipo de aplicaciones, sino también con un lenguaje de programación que cuente con especificaciones que lo conviertan en un lenguaje idóneo para el desarrollo de aplicaciones de este tipo.

2.2 HRT – HOOD COMO SISTEMA DE MODELAMIENTO ARQUITECTURAL DE LOS SISTEMAS DE TIEMPO REAL

En el desarrollo de sistemas de cualquier tipo, siempre se debe contar con una etapa de modelamiento, considerada una de las etapas más primordial de todas. Es en esta etapa donde se deben tener en cuenta detalles pormenorizados del sistema a implementar. Los Sistemas en Tiempo Real no escapan a este tipo de escenario, puesto que se requiere de un modelo consistente de diseño a implementar, el cual debe contar con requerimientos para Sistemas en Tiempo Real tales como “timing” (una buena regulación temporal) y “dependability” (seguridad en el funcionamiento).Los métodos o estándares típicos de diseño no brindan las previsiones adecuadas para expresar este tipo de requerimientos en los diseños.

Existen muchas técnicas y modelos de diseño, los cuales se pueden clasificar en informales, estructurados y/o formales(Burns,1995).Los métodos informales se caracterizan por el uso de lenguaje natural y diagramas imprecisos, la ventaja de esta notación es que es interpretada por un mayor grupo de personas, sin embargo, esta se puede prestar para dar un sin número de diferentes interpretaciones. Mientras que los métodos estructurados frecuentemente usan representaciones por medio de diagramas, los cuales son bien

definidos en comparación con los métodos informales, incluso los diagramas pueden contener representaciones sintácticas en diferentes lenguajes(Burns, 1995).Por otra parte los métodos formales se caracterizan por incluir operaciones matemáticas, lo cual aporta una clara ventaja a los diseños en cuanto a una descripción precisa de los requerimientos para desarrollo de la solución. Tal como se citaba en la introducción se puede decir que la desventaja de este método es que algunas veces se torna inmanejable de acuerdo a la complejidad de los sistemas a modelar, e incluso la notación no se considera fácilmente interpretable por personas poco familiarizadas con la misma.

La alta funcionalidad en los requerimientos de los Sistemas en Tiempo Real ha llevado a una búsqueda de un modelo que tenga la facilidad de interpretación de un sistema informal, pero la consistente capacidad gráfica de un sistema estructurado sin olvidar la precisión en la descripción de los detalles de un modelo formal (Burns,1995).

Existen hoy en día muchos métodos para Tiempo Real que buscan esta perfección, métodos tales como: MASCOT, JSD, HOOD, etc. Estos métodos a pesar de ser competitivos carecen de abstracción para Tiempo Real tal como la descripción de actividades periódicas y esporádicas, no son lo suficientemente fiables, sólo contemplan el tiempo de respuesta medio y no el peor, no se garantizan los requisitos temporales. En conclusión, esta clase de métodos de diseño no garantiza el cumplimiento de todos y cada uno de los requerimientos para un Sistema de Tiempo Real.

HRT-HOOD(*Hard Real-Time Hierarchical Object-Oriented Design*) fue desarrollado por Burns & Wellings en 1994. Es un método de diseño estructurado, basado en la descomposición jerárquica de objetos para sistemas de Tiempo Real, es derivado de HOOD (*Hierarchical Object-Oriented Design*) y esta basado en el estándar en la Agencia Europea del Espacio (ESA).Se puede decir que HRT-HOOD tiene la particularidad de ofrecer un sistema de especificación con representaciones gráficas y textuales, lo que permite representar las actividades periódicas y esporádicas, dándole una gran ventaja sobre los otros métodos, lo que contribuye a brindar un diseño consistente que cumple con las

especificaciones de los requerimientos. Sus Principios se basan en la abstracción, la descomposición jerárquica, el ocultamiento de información y el análisis temporal.

Una de las ventajas del uso de Hrt-Hood es que no solo permite descomponer el problema para lograr un minucioso análisis del mismo, sino que también logra definir objetos que pasan a formar parte de la solución del problema, ya sea a nivel de Software o Hardware. Descomponiendo la información de forma jerárquica, de manera que no exista ocultamiento de detalles ni inconvenientes por la abstracción del problema.

2.3 LENGUAJE ADA, COMO LENGUAJE PARA APLICACIONES EN TIEMPO REAL

Lenguaje de Programación Ada, es un lenguaje orientado a objetos con un fuerte tipado, aunque ha sido diseñado para ser un lenguaje de propósitos generales, cuenta con especificaciones para Tiempo Real que lo convierten en un lenguaje atractivo para este tipo de desarrollos. Además, permite un fácil traslado de componentes creados en Hrt-Hood al código del lenguaje Ada. Uno de los puntos fuertes que tiene Ada es su compilador, el Gnat. El Gnat entre muchas otras cosas soporta las especificaciones de la norma Posix, esta norma establece parámetros de sincronización y comunicación entre procesos para Tiempo Real y estos son soportados por librerías que contiene en Gnat como son la librería Gnull y la Gnatrl.

Hasta ahora podemos decir que un sistema en Tiempo Real es un sistema como cualquier otro, con la diferencia que este tipo de sistemas requieren de un funcionamiento adecuado dentro de uno parámetros temporales establecidos, los cuales se deben tener en cuenta en la fase de diseño. Mas adelante se mostrará en detalle cada una de las fases de diseño las cuales entre otras cosas involucran el uso de Hrt-Hood, además de tratar el tema de planificación temporal.

3. ANTECEDENTES

Los sistemas en Tiempo Real nacieron por la necesidad que tenia el hombre de llenar aquel vació que se producía en aquellos sistemas que requerían una arquitectura muy compacta y donde los procesos se deberían de llevar a cabo en ciertos rangos de tiempos estrictos.

Hoy en día los sistemas en Tiempo Real a nivel mundial han migrado de sistemas carentes de estrictas restricciones temporales a sistemas embebidos que cumplen de manera eficiente con aquellos objetivos para lo que fueron programados. Por tal motivo, se han desarrollado múltiples aplicaciones para Tiempo Real, entre las cuales se pueden mencionar sistemas de control para trenes, aviones e incluso con la ayuda del aonix que es un IDE (Sistema de Desarrollo)[3] para Ada, se desarrolló un trabajo para la estación espacial internacional .

También podemos encontrar Sistemas de Tiempo Real inmersos en sistemas un poco más complejos llamados sistemas distribuidos los cuales tienen procesos repartidos en distintas máquinas, e intercambian información mediante pasos de mensajes sobre algún sistema de comunicación, esta arquitectura se basa en el manejo de objetos y una de sus características es que permite comunicar máquinas con diferentes sistemas operativos o diferentes arquitecturas de diseño.

A continuación se presentan algunos ejemplos de proyectos y aplicaciones en donde se ha trabajado la problemática de Tiempo Real:

- TELORB[4]: Arquitectura para sistemas distribuidos en Tiempo Real, esta arquitectura es promocionada por Ericsson.
- CORBA[5]: Proporciona un entorno para el desarrollo y ejecución de aplicaciones distribuidas y en Tiempo Real por medio del manejo de objetos.

[3] htt://www.aonix.com
[4] http://www.telorb.com/index.html
[5] http://www.cs.wustl.edu/~schmidt/corba.html

- DESARROLLO DE CONTROL DE SISTEMAS DISTRIBUIDOS: Este proyecto se basa en el control de un sistema distribuido implementado en Lenguaje Ada, fue realizado por el Departamento de Ingeniería de Sistemas Telemáticos de la Universidad Politécnica de Madrid (Alonso, 2001).

4. OBJETIVOS

4.1 OBJETIVO GENERAL

Plantear un flujo de diseño para el desarrollo de aplicaciones de control de procesos continuos que requieran Tiempo Real, utilizando Ada como lenguaje de síntesis.

4.2 OBJETIVOS ESPECIFICOS

- Analizar las diferentes actividades que se presentan en el comportamiento de este Tipo de sistemas.
- Seleccionar los métodos y lenguajes más adecuados para representar estos sistemas.
- Implementar en lenguaje Ada los componentes que representan las actividades del sistema.
- Presentar modelos formales de cada uno de los componentes seleccionados.
- Presentar un procedimiento de diseño para este tipo de aplicaciones.
- Aplicar el procedimiento a la solución del control de nivel en un tanque.

5. JUSTIFICACIÓN

HRT-HOOD conforma una metodología de diseño y modelamiento contemporánea la cual se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas en Tiempo Real a construir. Lo cual proporciona un análisis confiable del sistema, lo que nos va a permitir reducir la cantidad de diferencias entre las características del prototipo y las especificaciones realizadas al principio de la fase de diseño, proporcionando a la fase de desarrollo una solución confiable y eficiente que cumple con los requisitos de Tempo Real que se ajustan al problema. El lenguaje de programación Ada, al ser concebido como un lenguaje para el desarrollo de aplicaciones en Tiempo Real, incluye la concurrencia y es totalmente orientado a objetos. Lo cual permite que se apliquen directamente los modelos actuales y reduce los errores presentes en el traslado del modelo a las herramientas de síntesis.

Dicho lenguaje cumple una parte importante en el papel del desarrollo de los Sistemas en Tiempo Real modernos, debido a su gran flexibilidad, alto tipado, gran velocidad de ejecución de las aplicaciones. Este lenguaje tiene en sí mismo un mecanismo de configuración de prioridades, las cuales son soportadas por el propio lenguaje y las bondades que ofrece el Gnat (Compilador del Lenguaje Ada), lo que nos va a permitir que el sistema operativo sobre el cual se monte la aplicación va a respetar el orden de las prioridades asignadas a las tareas que va a ejecutar la aplicación en Tempo Real.

Cuando se cuenta con metodologías de diseño tales como las expuestas anteriormente se puede contar con una solución al problema, que aparte de ser eficiente representa menos tiempo de diseño de lo que se tardaría con las metodologías tradicionales, reduciendo costos al utilizar menos los recursos con los que contamos, además de esto, en la fase de desarrollo de las aplicaciones para Tiempo Real implementadas por medio de lenguaje Ada requieren de menos introducción de líneas de código con llamados al sistema operativo, lo que nos representa menos tiempo en fase de desarrollo y por ende un menor costo por horas de programación. Otra ventaja es que el compilador de Ada, el Gnat al ser desarrollado bajo

licencia GPL, garantiza la libertad de compartir y modificar el compilador, lo que implica que para nuestros propósitos este software es gratis, lo cual representa un costo menos elevado de las aplicaciones que se programen en Ada. Las aplicaciones en Tiempo Real desarrolladas bajo parámetros confiables en cuanto a metodologías de diseño tales como HRT-HOOD e implementadas con lenguajes específicos como Ada, permiten a los usuarios finales contar con una solución robusta, escalable y eficientes al problema planteado lo que brinda una alta confiabilidad a la hora de utilizar la aplicación en sistemas críticos tales como sistemas de control industrial, comunicaciones, sistemas de control de vehículos, aviones o barcos donde se ponen en riesgo vidas humanas.

6. SISTEMAS EN TIEMPO REAL

6.1 ASPECTOS DE LA IMPLEMENTACIÓN DE LOS SISTEMAS EN TIEMPO REAL

Una de las características fundamentales de los Sistemas de Tiempo Real es la concurrencia. La implementación de un Sistema de Tiempo Real se basa en producir procesos o tareas que se ejecutan concurrentemente (Balbastre,2002). Se puede definir un proceso como un programa en ejecución, el cual incluye el valor actual del *program counter* (PC), registros, variables y posee un único hilo de control (Martínez, 2002).

Lo anterior significa que existe un pseudoparalelismo en la ejecución de la aplicación, a pesar de que existe una sola CPU real, ésta cambia periódicamente la ejecución de un proceso a otro, haciendo ver a los procesos como si estos se ejecutaran en forma paralela y que cada uno tuviese su propia CPU virtual. Aparte del estado de ejecución, los procesos pasan por varios estados, entre los cuales tenemos:

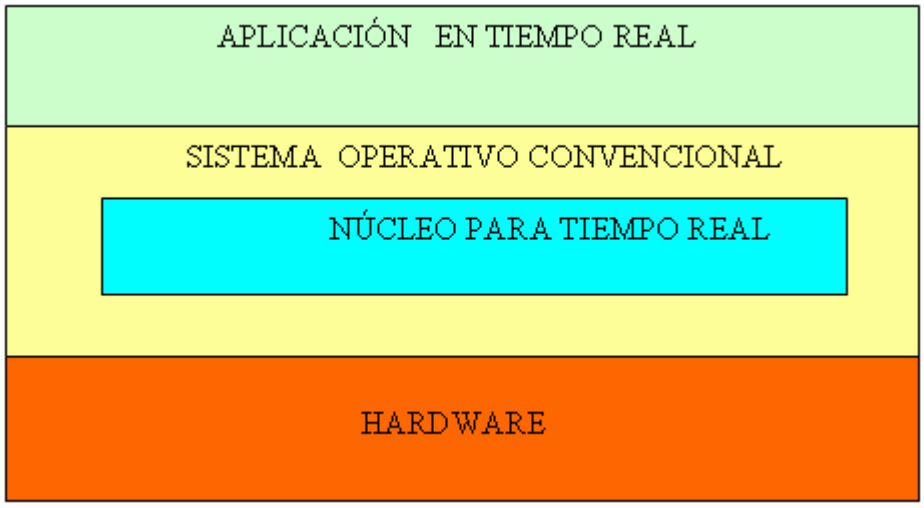
- Nuevo: Proceso que acaba de crearse y espera ser admitido para pasar al estado de listo
- Listo: Proceso que se encuentra en condiciones de ejecutarse pero que esta en espera de un turno para utilizar la CPU.
- Bloqueado: Proceso que no esta en condiciones de ejecutarse debido a la espera de que algún evento ocurra, por ejemplo la finalización de una operación de entrada o salida de datos.
- Terminado: Proceso que acaba de ejecutarse.

Cuando el sistema operativo entrega a la CPU un nuevo proceso, se produce un cambio de información, a este cambio se le conoce como cambio de contexto. En aplicaciones en Tiempo Real los sistemas operativos tienen un rol importante debido a que son ellos los que de alguna manera van a permitir los cambios de contexto de los procesos que debe ejecutar la aplicación en Tiempo Real.

Para garantizar que la aplicación sea predecible y cumpla con los plazos de cada actividad para Tiempo Real, se requiere de una plataforma que haga cumplir los procesos dentro de los tiempos establecidos, dicha plataforma debe contar con un sistema operativo para Tiempo Real como lo muestra la Figura 1. o se debe contar con un Sistema Operativo convencional sobre el cual se monta un núcleo para Tiempo Real que provea al sistema de la concurrencia adecuada, ver Figura 2.

Figura 1. Sistema Operativo Para Tiempo Real

Figura 2. Núcleo para Tiempo Real



Además de contar con un sistema operativo de Tiempo Real, se requiere de un análisis de la aplicación desarrollada para Tempo Real, con el fin de garantizar que está cumpla con sus plazos de ejecución, a este estudio se le denomina análisis de planificabilidad. También se debe recurrir a un lenguaje de programación confiable y eficaz que nos permita desarrollar una aplicación robusta y escalable que se caracterice por cumplir con los objetivos trazados desde el mismo diseño de la solución. Con la ayuda de metodologías orientadas a objetos, tales como HRT-HOOD, podremos realizar un buen diseño de la aplicación de acuerdo con el problema a tratar.

6.2 LENGUAJES DE PROGRAMACIÓN DE LOS SISTEMAS EN TIEMPO REAL

Las características de la programación de los Sistemas en Tiempo Real añade una complejidad adicional al diseño del software debido a:

- Necesidad de definir actividades concurrentes: el sistema de control debe asociar a cada bucle de control una tarea.
- Recursos compartidos.
- Gestión del tiempo: las activaciones de cada tarea se deben ejecutar en los instantes de tiempo que especifica su periodo y han de terminar antes de su plazo de entrega.
- Los sistemas han de ser fiables y robustos. Un fallo en la ejecución puede tener repercusiones importantes.
- La validación de los sistemas concurrentes requiere de herramientas específicas.

Históricamente se han utilizado un gran número de lenguajes de programación para el desarrollo de este tipo de sistemas. Se pueden encontrar aplicaciones realizadas en lenguaje Ensamblador, Fortran, C, Pascal, Modula-2, Ada, etc. También se definieron lenguajes específicamente para la programación de sistemas de Tiempo Real como CORAL 66, RTL/2 y PEARL entre otros, que se han utilizado de forma muy específica en entornos académicos. Sin embargo, no todos ellos son igualmente adecuados para el desarrollo de los Sistemas en Tiempo Real.

Las principales características que se deberían exigir serían:

- Concurrencia: abstracciones para la creación de tareas.
- Mecanismos de comunicación y sincronización entre tareas.
- Mecanismos para gestionar recursos compartidos.
- Acceso a bajo nivel.
- Gestión de dispositivos.
- Tolerancia a fallos: Manejo de situaciones excepcionales y erróneas.
- Transportabilidad: independencia del sistema operativo.
- Eficiencia.

6.2.1 Lenguajes Concurrentes. Los lenguajes concurrentes permiten incorporar las abstracciones necesarias para implementar aplicaciones concurrentes como construcciones del propio lenguaje (Modula-2, Ada, Java). Estas abstracciones permiten crear tareas, comunicarse y sincronizarse entre ellas. Estos lenguajes permiten independizar el desarrollo del sistema operativo sobre el que se ejecutan. Por ello, estas aplicaciones son transportables.

El Lenguaje Ada además de ser un lenguaje concurrente y de propósitos generales, ha sido diseñado específicamente para Sistemas de Tiempo Real empotrados, Ada a diferencia de lenguaje C, integra la concurrencia y el Tiempo Real (De La Puente, 2001). Es un lenguaje que permite la programación orientada a objetos, además de ser fuertemente tipado, permite la estructura en bloques, esta pensado para construir sistemas grandes, cambiantes y robustos, tiene la propiedad de permitir la construcción de módulos y esquemas genéricos, permite la extensión de tipos con herencia y presenta la posibilidad de hacer interfaces normalizadas con otros lenguajes, tales como C y Fortran.

Ada 95 es la versión más actual de Ada, la norma define unos anexos especializados para:

- Programación de Sistemas.
- Sistemas de Tiempo Real.
- Sistemas Distribuidos.
- Sistemas de Información.
- Cálculo Numérico.
- Fiabilidad y Seguridad.

Además los anexos definen:

- Paquetes de biblioteca.
- Mecanismos de implementación.

Entre los Lenguajes concurrentes podemos contar con el lenguaje de programación Java, el cual es un lenguaje pensado para construir Sistemas Distribuidos, se estructura sobre objetos dinámicos, presenta concurrencia integrada al lenguaje, incluye bibliotecas de clases (APIs) muy útiles, está pensado para que el código objeto sea portable. Es un lenguaje interpretado por una máquina virtual (JVM).

La definición original no es adecuada para Tiempo Real, debido a que la planificación de actividades concurrentes no está bien definida, además los mecanismos de sincronización son inadecuados e incluso la gestión dinámica de memoria introduce indeterminismo. La medida del tiempo no es suficientemente precisa y se han encontrado problemas con excepciones y concurrencia (De La Puente, 2001).

6.2.2 Lenguajes para Tiempo Real. Los lenguajes para Tiempo Real son lenguajes concurrentes que incorporan mecanismos para la gestión de las restricciones temporales características de los Sistemas de Tiempo Real y unos protocolos adecuados para la gestión de recursos compartidos. Podemos decir que existen distintos lenguajes de programación de Sistemas de Tiempo Real, Ada, Euclid RT, PEARL, Real-Time Fortran. De todos ellos Ada es el que tiene un carácter más general y dispone de un gran número de herramientas.

El lenguaje de programación Ada se diseñó a instancias del Departamento de Defensa de Los Estados Unidos con el fin de hacer un lenguaje común para el desarrollo de aplicaciones empotradas para reemplazar un gran número de aplicaciones desarrolladas en múltiples lenguajes distintos. Ada ofrece abstracciones para el diseño de aplicaciones de Tiempo Real que acceden al hardware e interrupciones. Además dispone de un modelo de gestión de excepciones que permite diseñar aplicaciones robustas y fiables.

Si la programación se realiza usando un lenguaje de Tiempo Real (Ada) y, puesto que las abstracciones de tareas están a nivel del lenguaje, el compilador genera las llamadas apropiadas al sistema operativo para hacer compatibles las abstracciones del lenguaje con las del sistema operativo, con lo cual se logra tener un código transportable. En cambio, si se ha usado un lenguaje secuencial (C), el programador tiene que incluir las llamadas al sistema de forma explicita en el código realizado.

6.3 SISTEMAS OPERATIVOS

Un Sistema Operativo es una parte importante de cualquier sistema de computación. Un sistema de computación puede dividirse en cuatro componentes: el *hardware*, el *Sistema Operativo*, los *programas de aplicación* y los *usuarios*. El hardware (Unidad Central de Procesamiento (UCP), memoria y dispositivos de entrada/salida (E/S)) proporciona los recursos de computación básicos. Los programas de aplicación (compiladores, sistemas de bases de datos, juegos de video y programas para negocios) definen la forma en que estos recursos se emplean para resolver los problemas de computación de los usuarios. (Instituto Tecnológico De Veracruz,2003)

También podemos decir que un Sistema Operativo, es un sistema que actúa como intermediario entre el usuario y el Hardware del Computador, con el objetivo principal de facilitar la comunicación y el objetivo secundario de permitir una utilización eficiente del Sistema de Computo (Barbazan, 2000).

Las funciones del Sistema Operativo son:

- La Gestión de Procesos.
- El Acceso a dispositivos de E/S.
- La Detección de errores y respuesta.
- La Contabilidad del sistema.

Los sistemas operativos los podemos clasificar en:

- Sistemas Operativos por lotes. Los Sistemas Operativos por lotes, procesan una gran cantidad de trabajos con poca o ninguna interacción entre los usuarios y los programas en ejecución. Se reúnen todos los trabajos comunes para realizarlos al mismo tiempo, evitando la espera de dos o más trabajos como sucede en el procesamiento en serie. Estos sistemas son de los más tradicionales y antiguos, y fueron introducidos alrededor de 1956 para aumentar la capacidad de procesamiento de los programas (Instituto Tecnológico De Veracruz,2003).

- Sistemas Operativos de Multitarea. Se distinguen por sus habilidades para poder soportar la ejecución de dos o más trabajos activos (que se están ejecutado) al mismo tiempo. Esto trae como resultado que la Unidad Central de Procesamiento (UCP) siempre tenga alguna tarea que ejecutar, aprovechando al máximo su utilización. Su objetivo es tener a varias tareas en la memoria principal, de manera que cada uno está usando el procesador, o un procesador distinto, es decir, involucra máquinas con más de una UCP. Sistemas Operativos como UNIX, Linux, Windows,MAC-OS, OS/2, soportan la multitarea(Instituto Tecnológico De Veracruz,2003).

- Sistemas Operativos de Tiempo Compartido**.** Permiten la simulación de que el sistema y sus recursos son todos para cada usuarios. El usuario hace una petición a la computadora, esta la procesa tan pronto como le es posible, y la respuesta aparecerá en la terminal del usuario. Los principales recursos del sistema, el procesador, la memoria, dispositivos de E/S, son continuamente utilizados entre los diversos usuarios, dando a cada usuario la ilusión de que tiene el sistema dedicado para sí mismo. Esto trae como consecuencia una gran carga de trabajo al Sistema Operativo, principalmente en la administración de memoria principal y secundaria. Ejemplos de Sistemas Operativos de tiempo compartido son Multics, OS/360 y DEC-10 (Instituto Tecnológico De Veracruz,2003).

- Sistemas Operativos Distribuidos. Permiten distribuir trabajos, tareas o procesos, entre un conjunto de procesadores. Puede ser que este conjunto de procesadores esté en un equipo o en diferentes, en este caso es trasparente para el usuario. Entre los diferentes Sistemas Operativos distribuidos que existen tenemos los siguientes: Sprite, Solaris-MC, Mach, Chorus, Spring, Amoeba, Taos, etc. (Instituto Tecnológico De Veracruz,2003).

- Sistemas Operativos de Red. Son aquellos sistemas que mantienen a dos o más computadoras unidas a través de algún medio de comunicación (físico o no), con el objetivo primordial de poder compartir los diferentes recursos y la información del sistema. El primer Sistema Operativo de red estaba enfocado a equipos con un procesador Motorola 68000, pasando posteriormente a procesadores Intel como Novell Netware. Los Sistemas Operativos de red mas ampliamente usados son: Novell Netware, Personal Netware, LAN Manager, Windows NT Server, UNIX, LANtastic(Instituto Tecnológico De Veracruz,2003).

- Sistemas Operativos de Tiempo Real**.** Los Sistemas Operativos de Tiempo Real son aquellos sistemas en los cuales tiene mucha relevancia los procesos y los instantes de tiempo en los cuales estos se ejecutan. Los sistemas de Tiempo Real deben ser predecibles y cumplir con ciertos parámetros que los conviertan en sistemas planificables como lo veremos mas adelante, se utilizan en entornos donde son procesados un gran número de sucesos o eventos.

6.3.1 Sistemas Operativos Para Tiempo Real. Muchos Sistemas Operativos de Tiempo Real son construidos para aplicaciones muy específicas como control de tráfico aéreo, bolsas de valores, control de refinerías, control de laminadores. También en el ramo automovilístico y de la electrónica de consumo, las aplicaciones de Tiempo Real están creciendo muy rápidamente. Otros campos de aplicación de los Sistemas Operativos de Tiempo Real son los siguientes:

- Control de trenes.
- Telecomunicaciones.
- Sistemas de fabricación integrada.
- Producción y distribución de energía eléctrica.
- Control de edificios.
- Sistemas multimedia.

Los Sistemas Operativos de Tiempo Real, cuentan con las siguientes características:

- Se utilizan en control industrial, conmutación telefónica, control de vuelo, simulaciones en Tiempo Real., aplicaciones militares, etc.
- Objetivo es proporcionar rápidos tiempos de respuesta.
- Procesa ráfagas de miles de interrupciones por segundo sin perder un solo suceso.
- Proceso se activa tras ocurrencia de suceso, mediante interrupción.
- Proceso de mayor prioridad expropia recursos.
- Por tanto generalmente se utiliza planificación expropiativa basada en prioridades.
- Gestión de memoria menos exigente que tiempo compartido, usualmente procesos son residentes permanentes en memoria.
- Poco movimiento de programas entre almacenamiento secundario y memoria.
- Gestión de archivos se orienta más a velocidad de acceso que a utilización eficiente del recurso (Instituto Tecnológico De Veracruz,2003).

Otro concepto importante es que un Sistema Operativo para Tiempo Real es un Sistema Operativo que debe responder a intervalos prefijados o ante eventos de una manera determinista y las tareas criticas deben recibir los recursos del sistema que necesiten y cuando los necesiten (Barbazan, 2000).

Los requisitos de un Sistema Operativo para Tiempo Real son:

- Determinismo
- Sensibilidad
- Control de Usuario
- Fiabilidad
- Tolerancia a los fallos

Gran parte de los Sistemas Operativos existentes adoptan el estándar POSIX.1b-1993 en lo que a prestaciones de Tiempo Real se refiere. Esencialmente, este estándar establece esquemas de diagramación priorizada, comunicación entre procesos mejorada y señalización de Tiempo Real (*real-time signals*), entre otros aspectos.

El acatamiento de estándares de este tipo hace a los sistemas tipo UNIX mucho más apropiados para las aplicaciones de Tiempo Real (Corvalán,2003).

Un ejemplo de Sistema Operativo conforme al mencionado estándar es QNX. La arquitectura de QNX se basa enteramente en un microkernel o micronúcleo del sistema operativo. Este microkernel implementa solamente cuatro servicios, a saber: planificación de tareas, comunicación entre procesos, comunicación de bajo nivel en red y despacho de interrupciones. Todos los demás servicios, tales como manejadores de dispositivos (*device drivers*) y sistemas de archivos (*file systems*) son implementadas como procesos de usuario cooperantes. Como resultado de este esquema, el kernel es suficientemente pequeño (aproximadamente 7 kilobytes de código) y rápido (Corvalán,2003).QNX también observa los estándares POSIX 1003.1 y POSIX 1003.2 relativos a interfaz de programa (*program interface*) y "shell" y utilidades (*shell and utilities*) respectivamente. Esto brinda comodidad a los programadores familiarizados con UNIX, ya que son proveídas todas sus prestaciones estándares, a saber: compiladores, depuradores, X-Windows y TCP/IP.

El desarrollo según un enfoque de microkernel presenta varias ventajas: La depuración de procesos de usuario es mucho más fácil que la de componentes de kernels grandes y la ejecución de los procesos de usuario en espacios de direccionamiento diferentes hace que errores en la gestión de memoria de módulos diferentes queden aislados. Otra ventaja adicional es la escalabilidad y la mayor sencillez en el mantenimiento y portado de sistemas basados en microkernels (Corvalán,2003).

En lo que a características de Tiempo Real se refiere, los microkernels proporcionan cambios de contexto mucho más rápidos. Más aun, procesos de usuario de Tiempo Real pueden expulsar a un manejador de dispositivo en cualquier momento. Finalmente, al ser muy pequeños los microkernels facilitan el cálculo de parámetros correspondientes a los peores casos de ejecución, entre los cuales podemos mencionar la latencia de interrupciones (Corvalán,2003).

En contrapartida, la principal debilidad de este tipo de sistemas es el nivel de desempeño alcanzable o *performance*. Una arquitectura basada en microkernel deposita gran parte de la carga en la comunicación entre procesos y cambios de contexto al proporcionar solamente los servicios más básicos. Otro ejemplo de sistema operativo para Tiempo Real es VxWorks. VxWorks esta basado en el esquema de huesped/objetivo (*host/target*).

Una máquina huésped tipo UNIX es utilizada para el desarrollo del software y para la ejecución de las porciones no-Tiempo-Real de la aplicación mientras que el kernel ejecuta las tareas de tiempo real en la máquina objetivo. Ambas máquinas se comunican utilizando TCP/IP. Aunque VxWorks no es compatible con UNIX, proporciona aquellas funciones relacionadas con las extensiones de Tiempo Real del estándar POSIX.1b. La mayoría de los API (*Application Program Interface*) son, sin embargo, propietarios. En VxWorks, el kernel(núcleo del sistema operativo) y las tareas corren en el mismo espacio de direccionamientos, alcanzándose así velocidades elevadas en los cambios de tarea (*tasks switching*).Un enlazador en tiempo de ejecución permite el cargado dinámico tanto de tareas como de módulos del sistema. Alguno otros ejemplos de Sistemas Operativos de Tiempo Real son: Solaris, Lyns OS y Spectra.

En síntesis, un sistema operativo para Tiempo Real es un sistema operativo capaz de garantizar los requisitos temporales de los procesos que controla. En contraposición a esto, los sistemas operativos convencionales no son apropiados para la realización de Sistemas de Tiempo Real, debido a que no tienen un comportamiento determinista y no permiten garantizar los tiempos de respuesta, esto por el hecho de tener ciertas características que impiden su uso como sistemas operativos para Tiempo Real, como por ejemplo, el uso de planificación para tiempo compartido lo que aseguran un uso equitativo del tiempo de CPU entre todos los procesos, además de contar con núcleos o kernels no desalojables haciendo que una llamada al sistema podría tardar demasiado tiempo para poder admitirlo en procesamiento de Tiempo Real(Corvalán,2003).

A pesar de los problemas mencionados, existe una forma de trabajar con sistemas operativos convencionales con el fin de utilizarlos como Sistemas Operativos para Tiempo Real. La idea es usar un núcleo para Tiempo Real tal como Open Ravenscar (http://www.openravenscar.com) o MarteOs (http://www.marte.unican.es), dichos núcleos o kernels introducen mejoras al Sistema Operativo convencional como es el caso de Linux, mejoras tales como la utilización de un planificador basado en prioridades y la introducción de sistemas de comunicación entre procesos (Corvalán,2003). Otra alternativa es el uso de RT-LINUX, el cual es un "parche" sobre el código de Linux, que además agrega módulos cargables al sistema operativo(Corvalán,2003).

RTlinux presenta las siguientes características:

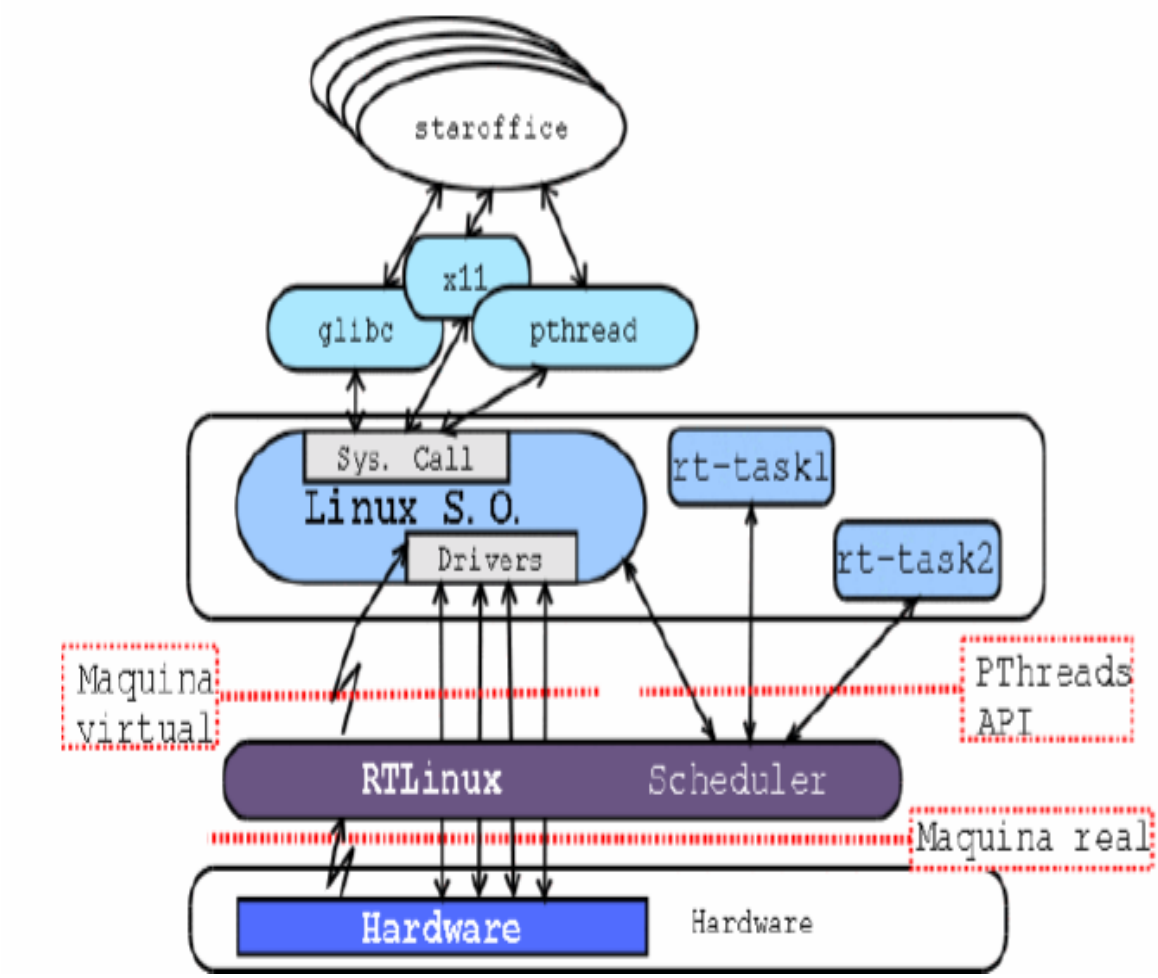
- A diferencia de otras aproximaciones para diseñar un S.O. de Tiempo Real, RTLinux no añade nuevas llamadas al sistema ni modifica ninguna de las ya existentes. Tampoco es una biblioteca para el programador.
- RTLinux se sitúa entre el hardware y el propio sistema operativo, creando una máquina virtual para que Linux pueda seguir funcionando.
- RTLinux toma el control de todas las interrupciones, e implementa un gestor de interrupciones por software.
- Las tareas RT-Linux se ejecutan utilizando el Run Time Support (RTS) de RTLinux.
- Las tareas de Tiempo Real (rt-task):

    - Comparten el mismo espacio de memoria que el núcleo, por lo que pueden acceder a todas las variables y funciones de éste.
    - No pueden hacer uso de las llamadas al sistema de Linux.
    - Se ejecutan en modo supervisor, esto es, pueden ejecutar cualquier instrucción de procesador y tienen acceso a todos los puertos de entrada/salida.

- Para poner en ejecución una rt-task se tiene que utilizar el sistema de módulos cargables de Linux.

- Los módulos son "trozos de sistema operativo" que se pueden insertar y extraer en tiempo de ejecución.

Figura 3. Rtlinux (Ripoll,2000)



En términos generales tanto los núcleos para Tiempo Real, como los Sistemas Operativos para Tiempo Real han adoptado el estándar POSIX.1b-1993 en lo que a prestaciones de Tiempo Real se refiere. Esencialmente, este estándar establece esquemas de diagramación priorizada, comunicación entre procesos mejorada y señalización de Tiempo Real (*real-time signals*), entre otros aspectos.

El acatamiento de estándares de este tipo hace a los sistemas tipo UNIX mucho más apropiados para las aplicaciones de Tiempo Real.(Ripoll,2000).La dependencia de los lenguajes que invocan de forma explícita las llamadas al sistema operativo (lenguajes secuénciales) y, en consecuencia, su poca transportabilidad, puede eliminarse utilizando una interfaz común para el acceso al sistema operativo. En este sentido POSIX (estándar 1003, Portable Operating System Interface basado en uniX) es un conjunto de normas IEEE/ ISO que definen interfaces de sistemas operativos y permiten desarrollar software portable y reutilizable (Martínez, 2002).

El objetivo de las extensiones de Tiempo Real, es añadir a POSIX básico los servicios que se necesitan para conseguir un comportamiento predecible y facilitar la programación concurrente. Estos servicios permiten:

- Planificación de procesos.
- Sincronización entre procesos.
- Compartición de memoria.
- Gestión de la memoria virtual.
- Señales para Tiempo Real.
- Definición de relojes y temporizadores.
- Colas de mensajes.
- Entrada/Salida síncrona y asíncrona.

Además, ofrece perfiles de aplicación:

- Sistema de Tiempo Real mínimo: sólo tareas (no procesos), sin gestión de memoria, ficheros ni terminal.
- Controlador de Tiempo Real: tiene sistema de ficheros y terminal.
- Sistema de Tiempo Real dedicado: tiene gestión de memoria y procesos pesados.
- Sistema de Tiempo Real generalizado: sistema completo con todo tipo de servicios.

Otro punto importante, es la tendencia hacia la utilización de Windows NT para el procesamiento en Tiempo Real. La razón principal es la compatibilidad con las versiones anteriores de Windows, además de la gran cantidad de aplicaciones populares y disponibles comercialmente. Esta tendencia se ve fortalecida por el deseo de adoptar un sistema operativo común tanto para las aplicaciones de oficina como para las de servicio o control en Tiempo Real, debido a que la existencia de un sistema operativo único reducirá considerablemente los costos derivados de la capacitación de personal. En contraposición a esta postura se debe mencionar que el kernel de Windows NT, en su formato actual, es incapaz de realizar procesamientos de Tiempo Real duro: el Win32 API simplemente no se encuentra diseñado para Tiempo Real, la notificación de una interrupción puede retrasarse por un intervalo de tiempo impredecible, y la preasignación de memoria puede ser problemática (Corvalán,2003).

Cabe anotar que para el desarrollo de este trabajo de grado se ha escogido el sistema operativo Linux RedHat 7.2 por su conocida estabilidad, además del núcleo para Tiempo Real RTLinux 3.2pre2 y la Aplicación RTLGnat 1.0 para portar código de lenguaje Ada sobre RTLinux.

7 FLUJO DE DISEÑO PARA LOS SISTEMAS DE TIEMPO REAL

7.1 PROCESOS DE DESARROLLO

Muchos métodos tradicionales de desarrollo de software incorporan procesos de desarrollo en los cuales podemos reconocer los siguientes aspectos:

- Definición de Requerimientos: En esta etapa se producen las especificaciones funcionales y no funcionales requeridas por el sistema.
- Diseño Arquitectural: Durante esta etapa se realiza una descripción del sistema a desarrollar.
- Diseño Detallado: En esta etapa se realiza un diseño completo del sistema.
- Codificación: En esta etapa se realiza la implementación del sistema diseñado.
- Prueba: Durante esta etapa se pone a prueba la eficacia del sistema.

Los métodos tradicionales y especialmente los aspectos tratados anteriormente resultan ineficientes para los Sistemas en Tiempo Real debido a que presentan significativas desventajas como problemas de especificaciones temporales que no son reconocidos durante la etapa de pruebas, o en el peor de los casos, durante la puesta en marcha del sistema, debido a estos inconvenientes se propone utilizar dos tipos de procesos de desarrollo que evitan los anteriores contratiempos (De La Puente, 2001).Estos procesos de desarrollo son el PROCESO DE DESARROLLO SECUENCIAL y EL PROCESO DE DESARROLLO ITERATIVO.

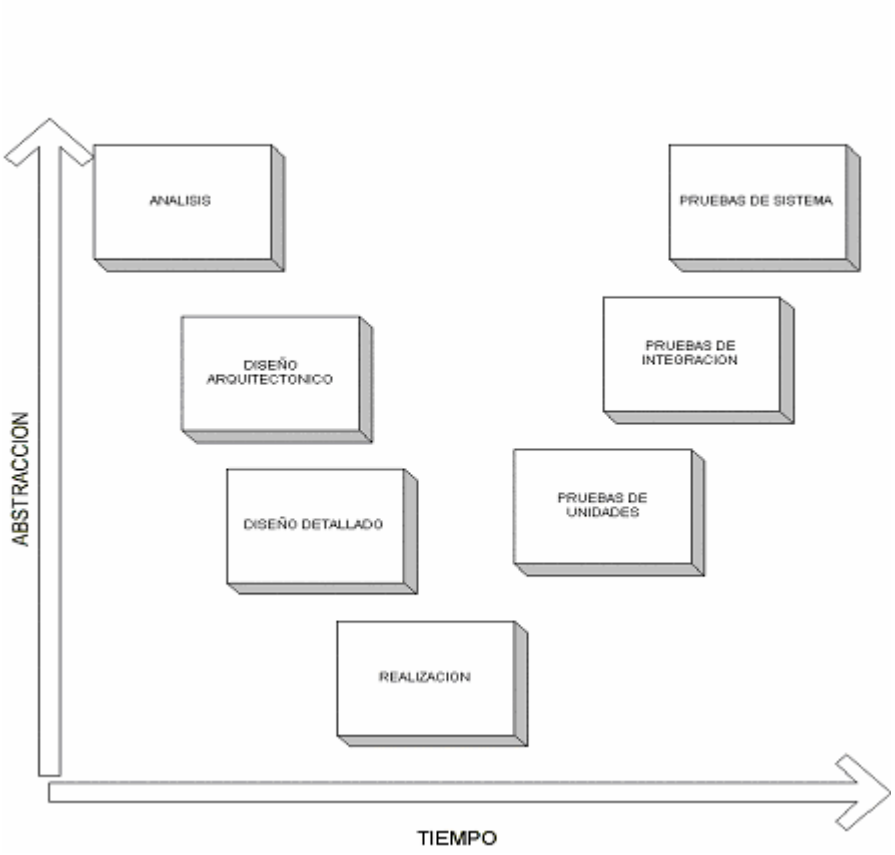
7.1.1. Proceso de Desarrollo Secuencial. El proceso de desarrollo secuencial es un proceso que presenta una secuencia de etapas tales como:

- Análisis: En esta etapa se realiza un estudio pormenorizado del sistema a implementar de acuerdo a los requerimientos exigidos por la solución.
- Diseño Arquitectónico: En esta etapa se realiza el diseño de la arquitectura que va a soportar la solución del problema a tratar, la arquitectura en el caso de Sistemas en Tiempo Real debe soportar una parte lógica y una parte física.

- Diseño Detallado: En esta etapa se trata de una forma minuciosa la manera de cómo se da solución a todos aquellos requerimientos de diseño que no le corresponden a la anterior etapa, tales como requerimientos temporales, etc.
- Realización: En esta etapa se produce la implementación del modelo diseñado en las anteriores etapas.
- Pruebas de Unidades: En esta etapa se prueba las unidades de manera independiente, para luego ser integradas.
- Pruebas de Integración: En esta etapa se prueba la integración del sistema con el propósito de verificar si las unidades siguen conservándose robustas después de haber sido integradas.
- Pruebas de Sistemas: En esta última etapa se prueba el sistema como tal con el fin de probar como se maneja en conjunto y si cumple con las expectativas de la solución que se planteo al hacer los diseños**.** (Burns, 1995)

Este proceso cuenta con otras características que resultan ventajosas, las cuales son la posibilidad de documentar cada etapa y realizar pruebas individuales o de unidades, además de pruebas de integración y pruebas del sistema al cabo de la realización del diseño. En la Figura 4. Podemos observar como se comporta la abstracción de este proceso a medida que aumenta el tiempo, es decir las etapas que están en la parte mas baja del gráfico presenta mas abstracción que las que están en la parte mas alta.

Figura 4. Proceso de Desarrollo Secuencial



7.1.2. Proceso de Desarrollo Iterativo. Debido a que los Sistemas en Tiempo Real están ligados a las restricciones impuestas por el entorno de ejecución, se debe tener un especial cuidado a la hora de diseñar cualquier tipo de sistema. La manera constructiva de describir el proceso de diseño de un determinado sistema y especialmente de un Sistema en Tiempo Real es realizando una progresión especifica de compromisos(Burns,1995).Estos compromisos o misiones definen las propiedades del sistema que esta en proceso de diseño.

Dado lo anterior, el proceso de desarrollo iterativo propone que a medida que se avanza en los niveles de abstracción se puedan crear propiedades que no se cambiarán las cuales se conocen como compromisos, mientras que las propiedades que se dejan para niveles inferiores se conocen como obligaciones.

A medida que el diseño se va depurando las obligaciones se convierten en compromisos debido a las restricciones impuestas por el entorno de ejecución. El entorno de ejecución puede imponer restricciones de recursos tales como: velocidad del procesador o también puede presentar restricciones de mecanismo, tales como: interrupción de prioridades, envío de tareas, etc.

Las obligaciones, los compromisos y las restricciones tienen una importante influencia en el diseño arquitectónico de cualquier aplicación. Por lo tanto, se han definido dos actividades del diseño arquitectural (Burns1995):

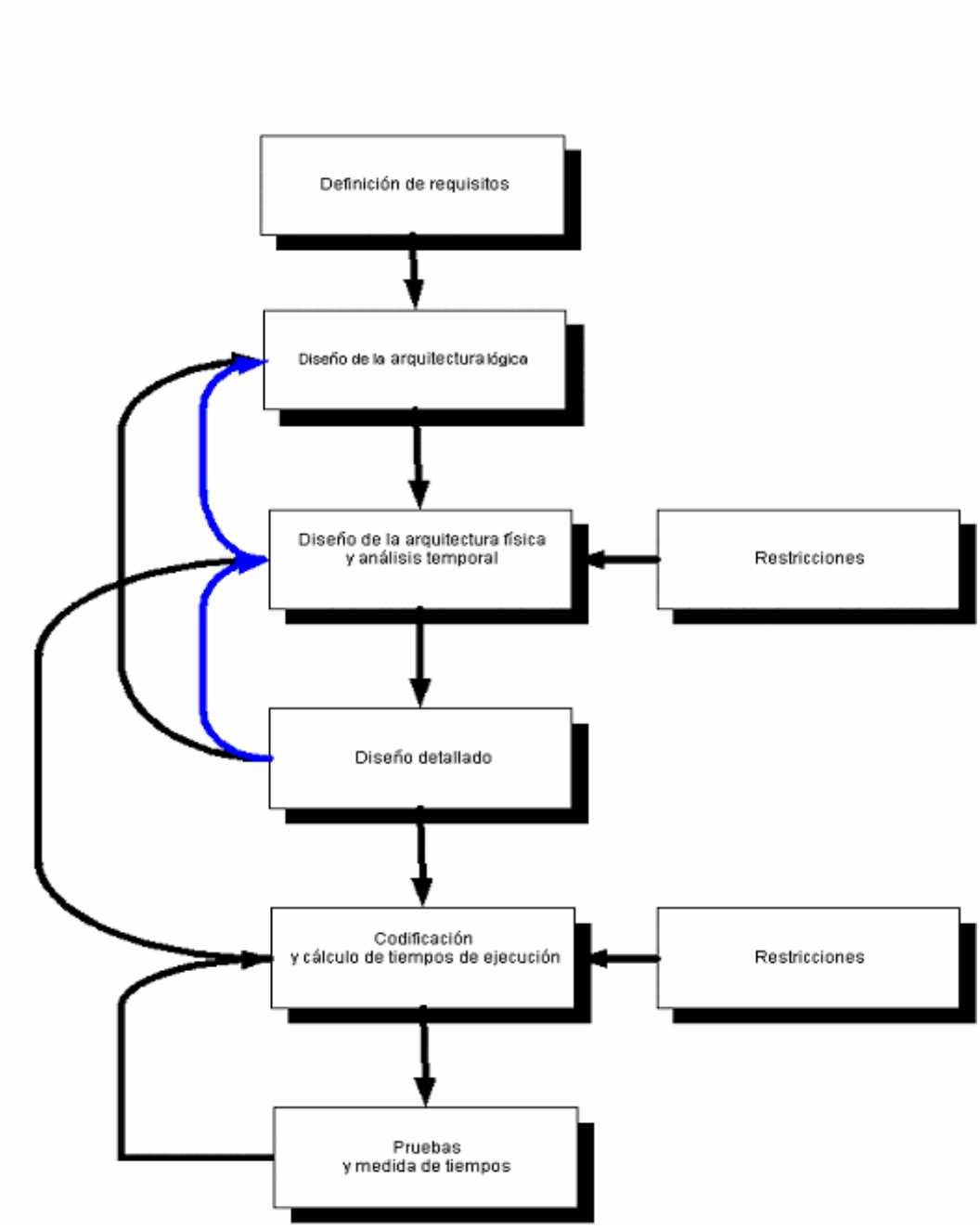
- Diseño de la arquitectura lógica.
- Diseño de la arquitectura física.

La arquitectura lógica incorpora compromisos que pueden ser independientes de las restricciones impuestas por el ambiente de ejecución, y se dirigen sobre todo satisfaciendo los requisitos funcionales (aunque la existencia de los requisitos temporales, tales como plazos, influenciará fuertemente la descomposición de la arquitectura lógica).

Por otro lado, la arquitectura física forma la base para afirmar que los requisitos no funcionales de la aplicación serán resueltos una vez realizados el diseño detallado y la implementación. Aunque la arquitectura física es un refinamiento de la arquitectura lógica su desarrollo será generalmente un proceso iterativo y concurrente en el cual ambos modelos son desarrollados y modificados. Una vez que las actividades del diseño tales como la arquitectura lógica y la física se han completado, el diseño detallado puede comenzar y el código para la aplicación puede ser producido. Cuando se ha alcanzado esto, el perfil de ejecución del código se debe estimar otra vez (analizando el peor caso para el tiempo de ejecución) con el fin de asegurarse que los tiempos de ejecución del peor caso han cumplido con las expectativas del diseño, es decir los tiempos deben ser exactos, si no lo son, el diseño detallado se deben revisar (si hay desviaciones pequeñas), o el diseñador

debe volver a las actividades de diseño arquitectónicas (si existen los problemas serios). Si la valoración indica que todo está bien, se procede la etapa de pruebas de la aplicación.

Figura 5. Proceso de Desarrollo Iterativo



La figura 5 nos muestra el proceso de desarrollo iterativo el cual ofrece ciertas características como son la presencia de un solo elemento en cada nivel de abstracción, además de contar con ciertas ventajas como lo son el hecho de presentar compromisos que implican propiedades que no se cambiarán y obligaciones que se dejan para niveles inferiores, la forma como esta planteado el proceso de desarrollo permite que las obligaciones se vayan transformando en compromisos, este proceso está sujeto a restricciones impuestas por el entorno de ejecución (De La Puente, 2001). El proceso de desarrollo iterativo también involucra el Diseño Arquitectural Lógico y el Diseño Arquitectural Físico.

Como se pudo observar, tanto el proceso de desarrollo Secuencial como el proceso de desarrollo Iterativo presentan grandes presentaciones a la hora desarrollar Sistemas de Tiempo Real lo cual brinda grandes ventajas en contraposición a los procesos de desarrollo convencionales. Para este trabajo de grado se ha escogido el proceso de desarrollo Iterativo debido a que este proceso de desarrollo presenta una estrecha relación con la metodología de diseño Hrt-Hood, metodología que hace parte de nuestra propuesta junto con el lenguaje de programación Ada.

7.2 DISEÑO ARQUITECTURAL LÓGICO

El diseño arquitectural lógico comprende todos aquellos detalles que están estrechamente ligados con la parte abstracta del diseño, incluyendo compromisos independientes del entorno de ejecución como por ejemplo: actividades periódicas y aperiódicas, etc. (Burns,1995).

Además, este tipo de diseño es ante todo una selección de objetos básicos los cuales no requieren de una mayor descomposición con todas sus interacciones completamente definidas. Los diseños realizados en Hrt-Hood componen la arquitectura lógica del sistema a implementar. Es por ello que Hrt-Hood juega un rol importante al permitir plasmar los detalles abstractos del sistema en una serie de objetos característicos en los sistemas de

Tiempo Real. Tales como: objetos PASIVOS, ACTIVOS, PROTEGIDOS, CICLICOS y ESPORADICOS.

7.3 NIVELES DE ABSTRACCION Y DISEÑO ORIENTADO A OBJETOS

Los métodos de diseño de software comprenden una serie de transformaciones desde los requisitos iniciales hasta el código ejecutable. Normalmente se consideran distintos niveles de abstracción en la descripción de un sistema:

- Especificación de requisitos
- Diseño arquitectónico
- Diseño detallado
- Codificación
- Pruebas

Cuanto más precisa sea la notación empleada en cada nivel mejor será la calidad del sistema final (Burns, 1995). Se resalta el hecho que HRT-HOOD permite combinar el uso de ventajas tales como el diseño orientado a objetos y la descomposición jerárquica.

- Abstracción, información oculta y encapsulación. Un objeto esta definido por el servicio que este provee a los usuarios; los detalles internos se encuentran ocultos. El servicio que se provee es especificado en la interfaz del objeto. Los detalles internos son descritos en los procesos de (OPCS) “Operation Control Structure” u operación de control estructurado, y este comportamiento se describe en un agente de sincronización llamado (OBCS) “Object Control Structure” o Estructura de control de objeto.

- Descomposición Jerárquica. Los objetos padres pueden ser descompuestos en objetos hijos. HRT-HOOD usa un termino llamado “incluir” para representar la relación padre hijo.

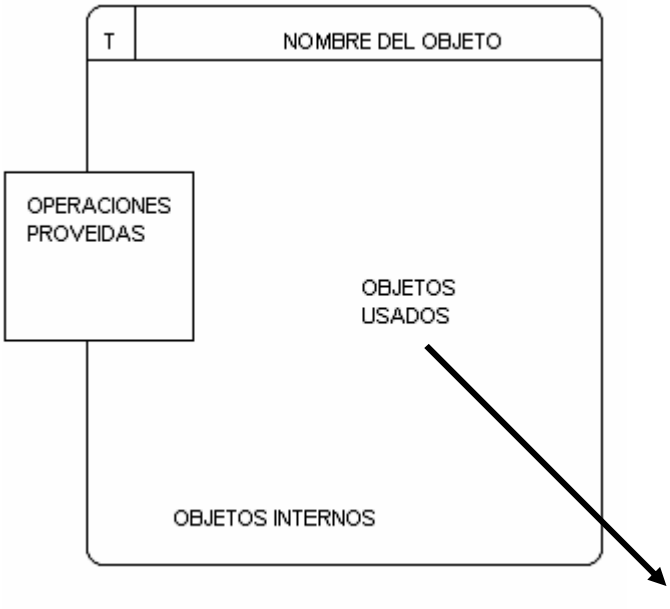
- Estructura de Control. Las operaciones en los objetos son activadas por flujos de control (hilos). En general puede haber varios hilos operando simultáneamente en un objeto. HRT-HOOD utiliza el termino “usa” para indicar que un objeto requiere el servicio de otro. El flujo de control puede ser secuencial o paralelo, dependiendo si el control es transferido directamente al objeto que requiere la operación o no.

- Flujo secuencial: Donde el control es transferido directamente al que requiere la operación. El flujo de control es descrito dentro de las operaciones internas (OPCS). Después del requerimiento, el control retorna al objeto original de donde fue tomado.(Burns, 1995).

- Flujo paralelo: El control no es transferido directamente al objeto llamado, pero un flujo independiente es activado en el (OBCS) del objeto llamado. Esta activación toma la forma de una solicitud de ejecución para una operación requerida. (Burns, 1995)

7.4 DESCRIPCIÓN DE OBJETOS EN HRT-HOOD

Figura 6. Modelo de Objeto

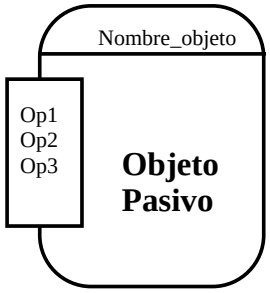


En La Figura 6 se observa como es el modelo de un objeto en Hrt-Hood, éste muestra varios aspectos, como lo son el nombre del objeto, el tipo del objeto, representado en la figura 6 con la letra T, las operaciones proveídas u operaciones que el mismo objeto va a desempeñar en la implementación, los objetos internos u objetos hijos del mismo, y por ultimo, los objetos que son invocados o usados en la eventualidad que el objeto requiera el uso de operaciones proveídas por otros objetos que no sean sus objetos hijos. HRT-HOOD facilita el diseño de la arquitectura lógica del sistema proporcionando los siguientes tipos de objetos:

- Objeto pasivo. No controla cuándo se ejecutan sus operaciones al ser éstas invocadas por otros objetos (es decir, no tiene restricciones de sincronización, como por ejemplo la exclusión mutua que garantizan los procedimientos de una entidad protegida o condiciones de sincronización) (Martínez, 2002). No invoca operaciones en otros

objetos de forma espontánea (es decir, no tiene tareas).Un objeto pasivo no cuenta con OBCS "Object Control Structure"y no tiene hilos. El OBCS es un agente de sincronización de los objetos, este se encarga de sincronizar el flujo de control entre objetos cuando un objeto es llamado por otro objeto o cuando un objeto usa a otro objeto. (Burns, 1995)

Figura 7. Objeto Pasivo



- Objeto activo. Es la clase mas general de objeto, sus operaciones pueden ser restringidas o no. Las operaciones que no son restringidas se ejecutan tan pronto son solicitadas, similar a las operaciones en un objeto pasivo, este tipo de operaciones se utilizan para acceder a un objeto con el propósito de leer la información que contiene o compartir datos no protegidos (Martínez, 2002).

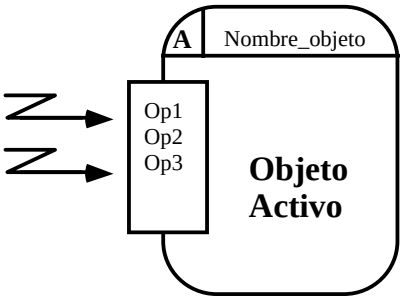
Las operaciones restringidas de un objeto activo se realizan bajo el control del OBCS, además existen dos clases de restricciones, las cuales pueden afectar al objeto cuando la solicitud de la operación es ejecutada, lo cual tiene efecto sobre el objeto llamador:
La primera es la activación de restricciones, la cual impone limitaciones sobre la solicitud de una operación de determinado objeto acorde con el estado interno de este. Una operación se dice que esta abierta si el estado interno del mismo objeto permite la ejecución de dicha operación, por ejemplo si se quiere escribir en un buffer controlado por un objeto, este dejara que se realice la operación si el mismo no esta lleno, pero de lo contrario se activará la restricción que impida la solicitud de la operación de escritura sobre el buffer lo que nos permite decir que la operación esta cerrada.

La segunda nos muestra los Tipos de solicitudes restringidas, las cuales indican el efecto sobre el objeto llamador que solicita una operación a otro objeto. Los siguientes tipos de solicitudes son soportadas:

- *Asynchronous Execution Request* (ASER):
  Cuando una operación ASER es llamada, el objeto llamador no es bloqueado por la solicitud.
- *Loosely Synchronous Execution Reques* (LSER): Cuando una operación LSER es llamada, el llamador es bloqueado por la solicitud hasta que el objeto llamado este listo para atender la solicitud.
- *Highly Synchronous Execution Request* (HSER). Cuando una operación HSER es llamada, el llamador es bloqueado por la solicitud hasta que el objeto llamado haya servido la solicitud.

También podemos decir que los objetos activos pueden invocar operaciones en otros objetos de forma espontánea y presentan actividad propia.

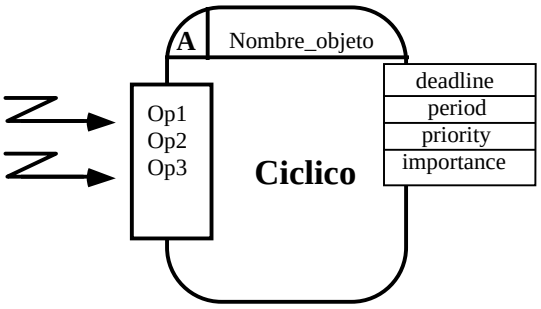
Figura 8. Objeto Activo



- Objeto protegido. Deben ser analizables, ya que imponen bloqueo a sus invocantes. Puede controlar cuándo se ejecutan sus operaciones al ser éstas invocadas por otros objetos. Se permite al diseñador restringir las condiciones de sincronización del objeto. No puede invocar operaciones en otros objetos de forma espontánea(Martínez, 2002).
  Este tipo de objeto es usado para controlar el acceso a recursos que son utilizados por objetos que funcionan en Tiempo Real. Los objetos protegidos no utilizan OBCS.

- Objeto cíclico. Objeto que representa una actividad periódica. Puede invocar operaciones en otros objetos de forma espontánea. Se considera un objeto activo en el sentido de que tiene sus propios hilos independientes de control. En general los objetos cíclicos pueden comunicarse y sincronizarse con otros hilos para Tiempo Real por llamadas a operaciones en objetos protegidos. Varias operaciones restringidas son habilitadas para los objetos cíclicos, estas cuando son abiertas requieren de una respuesta rápida por parte del hilo del objeto. El OBCS de un objeto cíclico interactúa con el hilo para forzar una transferencia de control asíncrona. La principal operación habilitada es la Solicitud de Transferencia Asíncrona de Control "Asynchronous, asynchronous transfer of control request" (ASATC). (Burns, 1995). Esta operación es similar a la ASER para objetos activos excepto que esta demanda que el objeto cíclico reaccioné "inmediatamente". La solicitud resultará en una transferencia asíncrona de control en el hilo del objeto cíclico. En este caso la llamada es también asíncrona y el llamador no es bloqueado esperando a que la transferencia de control tome lugar.

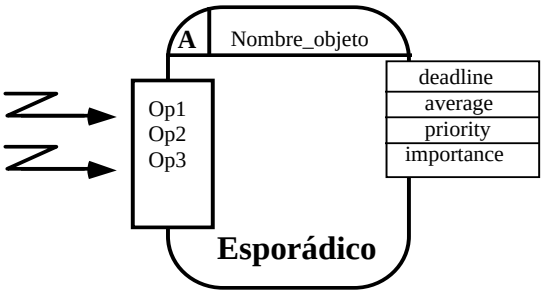
Figura 9. Objeto Cíclico



- Objeto esporádico. Objeto que representa una actividad esporádica. Puede invocar operaciones en otros objetos de forma espontánea. Tiene una *única* operación, que es la que lo invoca. Los objetos esporádicos son considerados objetos activos en el sentido que ellos tienen sus propios hilos de control independientes. Cada objeto esporádico tiene una operación restrictiva simple, usualmente llamada START, la cual es llamada para invocar la ejecución de un hilo.

Esta operación no es del tipo de las que bloquean al llamador (ASER), esta puede ser llamada por una interrupción. Cada objeto esporádico tiene un hilo y un OBCS(Burns,1995).

Figura 10. Objeto Esporádico



En general podemos decir que:

- HRT-HOOD distingue entre la sincronización requerida para ejecutar una operación de un objeto y cualquier actividad interna concurrente e independiente dentro del mismo objeto. El agente de sincronización de un objeto es llamado Estructura de Control de Objeto (OBCS) “Object Control Structure”. La actividad concurrente dentro del mismo objeto es llamada hilo del objeto. Los hilos se pueden ejecutar de manera independiente a las operaciones, pero cuando el hilo ejecuta una operación, el orden de ejecución es controlado por el OBCS.
- Las actividades cíclicas y/o esporádicas son comunes en los Sistemas de Tiempo Real, cada una contiene un hilo que es planificado en el run-time.

- Los objetos protegidos controlan el acceso a la información que es compartida por los hilos de un objeto (objetos cíclicos o esporádicos); en particular ellos proporcionan la exclusión mutua (Burns, 1995).

7.5 OPERACIONES DE LOS OBJETOS

- Objetos pasivos. Sus operaciones se ejecutan en cuanto se invocan.

- Objetos activos. Sus operaciones pueden bloquearse por restricciones de invocación asíncrona, síncrona, invocación remota.

- Objetos protegidos. Sus operaciones se ejecutan en exclusión mutua. La invocación puede ser asíncrona o síncrona, o pueden tener temporizaciones

- Objetos cíclicos. Sus operaciones se ejecutan a intervalos regulares.

- Objetos esporádicos. Tienen una única operación asíncrona (START), la cual puede estar ligada a una interrupción. Pueden tener operaciones de transferencia de control asíncrona.

7.6 RELACIONES DE USO EN HRT-HOOD

- Tanto en HOOD como en HRT-HOOD, los objetos pasivos no pueden usar operaciones de otros objetos diferentes a los de su clase, en cambio los objetos activos tienen la libertad de usar operaciones de otros objetos.

- Los objetos cíclicos y los esporádicos no deben usar operaciones restrictivas de objetos activos a menos que estas sean asíncronas (ASER).

- Los objetos cíclicos y los esporádicos pueden usar operaciones restringidas de un objeto protegido.

- Los objetos cíclicos y esporádicos pueden llamar las operaciones restringidas de otros objetos cíclicos y esporádicos.

- Los objetos protegidos pueden usar únicamente operaciones restringidas asíncronas (ASER) de otros objetos, o utilizar operaciones restringidas de objetos protegidos.

La siguiente tabla resume la relación de uso de HRT-HOOD:

Tabla 1. Relaciones de Uso

LLAMADO ---------- LLAMADOR	**ACTIVO**	**CICLICO**	**ESPORADICO**	**PROTEGIDO**	**PASIVO**
ACTIVO	SI	SI	SI	SI	SI
CICLICO	SOLO ASINCRONO	SI	SI	SI	SI
ESPORÁDICO	SOLO ASINCRONO	SI	SI	SI	SI
PROTEGIDO	SOLO ASINCRONO	SOLO ASATC	SOLO ASATC/START	SI	SI
PASIVO	NO	NO	NO	NO	SI

7.7 REGLAS DE INCLUSIÓN (DESCOMPOSICIÓN) EN HRT-HOOD

Las relaciones de inclusión para HRT-HOOD son las siguientes:

- Un objeto activo puede incluir todos los tipos de objeto que existen.
- Un objeto pasivo solo puede incluir otros objetos pasivos.
- Un objeto protegido pude incluir objetos pasivos, y un objeto protegido.

  La intención es que los objetos protegidos no tengan hilos de control separados. Consecuentemente ellos no podrán descomponerse en objetos que tengan sus propios hilos de control. Además, un objeto protegido padre garantiza que sus datos serán accesados bajo exclusión mutua.

- Un objeto esporádico puede incluir al menos un objeto esporádico con uno o más objetos pasivos y protegidos.
- Un objeto cíclico puede incluir al menos un objeto cíclico con uno o más objetos pasivos, esporádicos o protegidos.

Las reglas de inclusión son resumidas en la siguiente tabla.

Tabla 2. Reglas de Inclusión

HIJO **----------** **PADRE**	**ACTIVO**	**CICLICO**	**ESPORADICO**	**PROTEGIDO**	**PASIVO**
ACTIVO	SI	SI	SI	SI	SI
CICLICO	NO	SI	SI	SI	SI
ESPORADICO	NO	SI	SI	SI	SI
PROTEGIDO	NO	NO	NO	SI	SI
PASIVO	NO	NO	NO	NO	SI

7.8 DISEÑO ARQUITECTURAL FÍSICO

El diseño arquitectural físico se considera como el mapeo del diseño arquitectural lógico, en una fuente de recursos físicos que permitan lograr el objetivo del sistema. Para garantizar un diseño sólido se debe contar con un modelo de diseño que facilite el análisis del sistema en general, además de contar con un comportamiento predecible por parte del sistema operativo con el fin de garantizar el cumplimiento temporal de la aplicación.

Como el diseño arquitectural lógico y el físico deben ir de la mano, en el diseño arquitectural físico se deben considerar los siguientes parámetros:

Sistema de objetos terminales con atributos temporales:

- Cíclicos: período, plazo, tiempo de cómputo.
- Esporádicos: separación mínima entre activaciones consecutivas, plazo, tiempo de cómputo.
- Protegidos: tiempo de cómputo.
- Pasivos: tiempo de cómputo.

Modelo de ejecución analizable:

- Monoprocesador.
- Planificación con prioridades fijas y expulsión.
- Exclusión mutua con techo de prioridad inmediato.

Análisis temporal:

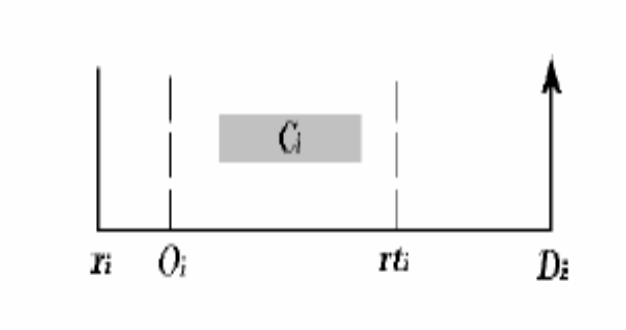
- Cálculo de tiempos de respuesta.

Cabe anotar que HRT-HOOD hace parte integral de la filosofía de programación orientada a objetos lo que permite mejoras de amplio alcance en la forma de diseño, desarrollo y posterior mantenimiento del software. Dada la capacidad que tiene HRT- HOOD para representar la abstracción de los modelos de software y hardware por medio de objetos, se le brinda al diseñador la posibilidad de modelar una solución al problema planteado de una forma simple, aunque sin carecer de una buena validación que cumpla con las expectativas del diseño.

7.9 PLANIFICACION DE SISTEMAS EN TIEMPO REAL

Siguiendo el esquema planteado por el proceso de desarrollo iterativo y de acuerdo a lo expresado por el diseño arquitectural físico, vamos a tratar el tema de la planificación de sistemas en Tiempo Real. De acuerdo a la definición de un sistema de Tiempo Real expresada con anterioridad, podemos decir que para garantizar el buen desempeño de un sistema de estas características, es necesario realizar un análisis que nos permita verificar si los plazos impuestos por cada una de las tareas será cumplido. Los parámetros temporales que rigen la activación de una tarea son los siguientes:

Figura 11.Parámetros Temporales(Balbastre,2002)



- *Release time* (ri): Es el instante en el cual la activación de la tarea está preparada para ejecutarse.

- Tiempo de cómputo (Ci): Es el tiempo que tarda el procesador en ejecutar las acciones que tiene asignadas la tarea.

- Plazo de entrega (Di): Es el instante en el cual la activación de la tarea debe haber terminado.

- Tiempo de respuesta (rti) Es el instante (relativo al *release time*) en el que la tarea finaliza su ejecución.

- Desplazamiento (Oi): En algunos casos, la aplicación de la tarea no empieza en el instante que llega el *release time*. Por lo tanto se produce un desfase o desplazamiento en su instante de inicio de ejecución.
- Periodo de la tarea (Pi): Frecuencia con la cual se repite la tarea.

Un sistema se considera planificable cuando la activación de la tarea concluye dentro del plazo especificado, es decir el tiempo de respuesta de la tarea (rti) debe ser menor o igual al plazo de entrega(Di) (Balbastre,2002).

7.9.1 Algoritmos De Planificación. En un sistema de Tiempo Real la función del algoritmo de planificación es determinar un orden de ejecución de las tareas. En el diseño de un sistema de Tiempo Real la elección de la política de planificación depende de varios puntos: número de procesadores, relaciones de precedencia entre tareas, dinámica del sistema a controlar, etc. Los algoritmos de planificación son métodos que permiten al diseñador comprobar si el sistema cumple o no con las restricciones temporales que se le imponen (Balbastre, 2002).

Los algoritmos de planificación de Tiempo Real pueden clasificarse también como estáticos o dinámicos. Un algoritmo de planificación estático es aquel cuya planificación se calcula fuera de línea. Un algoritmo de planificación dinámica determina el orden de ejecución de las tareas durante la ejecución del sistema.

"Los planificadores estáticos disponen la ejecución de las tareas en una determinada secuencia, la cual se repite cíclicamente. A esta técnica se la conoce como ejecutivo cíclico"[6]. La planificación estática se caracteriza por un bajo coste de *run-time*.

[6] ZAMORANO, J. Planificación Estática De Procesos En Sistemas De Tiempo Real Críticos, citado por BALBASTRE, Patricia. Modelo De Tareas Para La Integración Del Control y La Planificación En Sistemas De Tiempo Real. Valencia, 2002 ,165 p. Tesis Doctoral (Ingeniero Electrónico). Universidad Politécnica de Valencia. Departamento de Informática de Sistemas y Computadores.

Sin embargo, es poco flexible y requiere una completa predecibilidad. Razones por las cuales actualmente no tienen mucha aplicación en el desarrollo de sistemas de Tiempo Real.

Los algoritmos de planificación dinámica, por el contrario, se adaptan fácilmente a cambios en tiempo de ejecución. Dentro de los planificadores dinámicos, existe una familia de planificadores llamados por prioridades. Con frecuencia estos son los algoritmos que se utilizan para asignar el orden de ejecución de las tareas en un sistema de Tiempo Real.

7.9.2 Planificadores basados en Prioridades. Con este tipo de planificadores cada tarea tiene asociado un valor de prioridad, o de importancia, de forma que siempre se ejecuta la tarea activa de mayor prioridad.

Los planificadores por prioridades pueden subdividirse en:

- Planificadores por prioridades fijas. En los cuales se asigna un valor de prioridad a cada tarea al inicio de la ejecución del sistema, y este se conserva durante todo el tiempo de funcionamiento de la aplicación.
- Planificadores por prioridades dinámicas. Si la prioridad de cada tarea cambia durante la ejecución, entonces se dice que el planificador es por prioridades dinámicas.

Dentro de los planificadores por prioridades fijas los más conocidos son el *Rate Monotonic* (RM), según Liu[7] , dado que asigna la mayor prioridad a la tarea con el periodo (Pi) más corto, y el *Deadline Monotonic* (DM), según Leung[8], donde la tarea más prioritaria es la que tiene el menor Plazo de entrega o *deadline*(Di).

---

[7] LIU, C.L. y Layland, J.W. Scheduling Algorithms For Multiprogramming In A Hard Real-Time Environment, citado por BALBASTRE, Patricia. Modelo De Tareas Para La Integración Del Control y La Planificación En Sistemas De Tiempo Real. Valencia, 2002 ,165 p. Tesis Doctoral (Ingeniero Electrónico). Universidad Politécnica de Valencia. Departamento de Informática de Sistemas y Computadores.

Respecto a prioridades dinámicas, los algoritmos más ampliamente estudiados son el EDF (Earliest Deadline First), tal como lo dice Liu[9] , debido que prioriza según el *deadline* absoluto y el LLF (*Leas*t *Laxity First*), según Mok[10], el cual que asigna la mayor prioridad a la tarea con menor holgura. Sin embargo, el algoritmo EDF, a diferencia de los algoritmos con prioridades fijas, no puede garantizar el cumplimiento de los plazos de tareas críticas en presencia de sobrecargas. Razón por la cual en sistemas críticos se acostumbra utilizar algoritmos con prioridades estáticas.

Como veremos mas adelante, con el uso de las herramientas propuestas en este trabajo es posible asignar las prioridades de cada tarea al inicio de la ejecución del sistema y conservar las mismas durante todo el tiempo de ejecución de la aplicación.

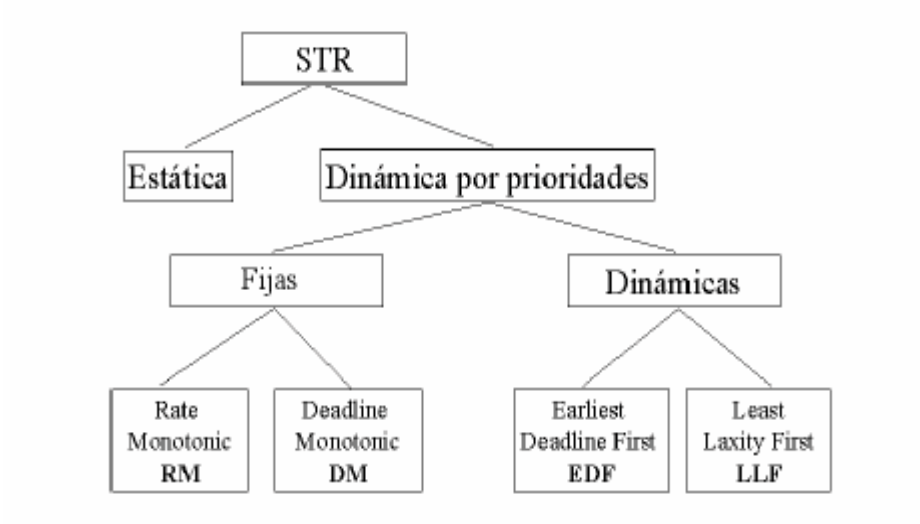
---

[8] LEUNG, J.y WHITEHEAD, J. On The Complexity Of Fixed-Priority Schedulings Of Periodic, Real-Time Tasks. Performance Evaluation, citado por BALBASTRE, Patricia. Modelo De Tareas Para La Integración Del Control y La Planificación En Sistemas De Tiempo Real. Valencia, 2002 ,165 p. Tesis Doctoral (Ingeniero Electrónico). Universidad Politécnica de Valencia. Departamento de Informática de Sistemas y Computadores.

[9] LIU, C.L. y Layland, J.W. Scheduling Algorithms For Multiprogramming In A Hard Real-Time Environment, citado por BALBASTRE, Patricia. Modelo De Tareas Para La Integración Del Control y La Planificación En Sistemas De Tiempo Real. Valencia, 2002 ,165 p. Tesis Doctoral (Ingeniero Electrónico). Universidad Politécnica de Valencia. Departamento de Informática de Sistemas y Computadores.

[10] MOK, A. y DERTOUZOS, M. Multiprocessor Scheduling In A Hard Real-Time Environment, citado por BALBASTRE, Patricia. Modelo De Tareas Para La Integración Del Control y La Planificación En Sistemas De Tiempo Real. Valencia, 2002 ,165 p. Tesis Doctoral (Ingeniero Electrónico). Universidad Politécnica de Valencia. Departamento de Informática de Sistemas y Computadores.

Figura 12. Sistema de Tiempo Real (Balbastre,2002)



7.9.3 Planificador de Prioridades fijas. El algoritmo de planificación RM supone que las tareas tienen el plazo igual al periodo (Di = Pi). Un test de planificabilidad suficiente para el RM es el proporcionado por Liu[11], donde expresa que un conjunto de tareas T es planificable bajo RM si:

$$UT \leq n\left(2^{1/n} - 1\right)$$

Siendo $UT = \sum_{i=1}^{n} ci / pi$ el factor de utilización del procesador y n el número de tareas.

Entre tanto, Lehoczky[12] expresa que un conjunto de tareas T es planificable bajo el DM si y sólo si:

[11] LIU, C.L. y Layland, J.W. Scheduling Algorithms For Multiprogramming In A Hard Real-Time Environment, citado por BALBASTRE, Patricia. Modelo De Tareas Para La Integración Del Control y La Planificación En Sistemas De Tiempo Real. Valencia, 2002 ,165 p. Tesis Doctoral (Ingeniero Electrónico). Universidad Politécnica de Valencia. Departamento de Informática de Sistemas y Computadores.
[12] LEHOCZKY, J. The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behaviour, citado por BALBASTRE, Patricia. Modelo De Tareas Para La Integración Del Control y La Planificación En Sistemas De Tiempo Real. Valencia, 2002 ,165 p. Tesis Doctoral (Ingeniero Electrónico). Universidad Politécnica de Valencia. Departamento de Informática de Sistemas y Computadores.

$$\forall 1 \leq i \leq n \qquad \min_{t \leq 0 \leq Di} \qquad \sum_{\forall j \mathrm{E} hp(i)} \frac{Ci}{t} \left\lceil \frac{t}{Pj} \right\rceil$$

7.9.4 Planificador de Prioridades Dinámicas. Los algoritmos de planificación con prioridades dinámicas utilizan información actualizada dinámicamente sobre los parámetros temporales de las tareas. A partir de esta información, deciden con qué prioridad se debe ejecutar la tarea en ese instante. El algoritmo más extendido de prioridades dinámicas es *Earliest Deadline First* (EDF), que elige para ejecutar en cada momento la tarea cuyo plazo de terminación (absoluto) está más próximo.

La principal ventaja es que el límite de utilización para un sistema con tareas periódicas e independientes es del 100%. Dicho de otra forma, la condición de planificabilidad de un sistema planificado con EDF es la siguiente:

$$\sum Ci/Pi < 1$$

7.9.5 Análisis según la Utilización En principio, podría pensarse que la condición para que un sistema con *n* tareas sea planificable, es decir, que todas las tareas cumplan sus plazos, es que éstas no requieran más tiempo de cómputo del disponible. Ello puede representarse por medio de la siguiente inecuación:

$$\sum Ci/Pi < 1$$

Al cociente *Ci/Pi* se le llama *utilización* de la tarea i. A la suma de las utilizaciones de todas las tareas se le llama *utilización del sistem*a. Lo que esta inecuación representa es la condición de que la capacidad de proceso solicitada por todas las tareas por unidad de tiempo (la utilización del sistema) no supere el 100% del tiempo.

Sin embargo este límite de utilización sólo se puede alcanzar en algunos casos: cuando la ejecución de todas las tareas encaja perfectamente en todas las posibles fases, esto es, los periodos de las tareas son uniformemente armónicos. Un sistema es *uniformemente armónico* si el periodo de cada tarea es un divisor exacto de los periodos más grandes. Se pueden dar casos en los cuales un sistema puede perder sus plazos con una utilización inferior al 100% lo que comprueba que un porcentaje de utilización del 100% no es garantía para que se cumplan los plazos.

En 1973, Liu y Layland formularon la *condición suficiente de planificabilidad* de un sistema de Tiempo Real con las restricciones de RM, es decir: planificador expulsivo, con asignación de mayor prioridad a las tareas más frecuentes, tareas periódicas e independientes y plazos iguales a los periodos. Para un sistema con *n* tareas, numeradas de 1 hasta n, esta condición es la siguiente:

Si $n * [(2 \wedge (1/n)) - 1] \geq \sum Ci/Ti$, el sistema es planificable

Ello no significa que si un sistema tiene una mayor utilización no pueda ser planificable, sino que es suficiente comprobar la inecuación anterior para asegurar que el sistema es planificable bajo RM. Así pues, esta inecuación representa una condición *suficiente pero no necesaria* de planificabilidad (Martínez,2002).

Considérese un sistema con las tres tareas de la tabla siguiente, en el que la utilización total es del 100%. La figura muestra el diagrama temporal de su ejecución y, como puede comprobarse, todas las tareas cumplen sus plazos (la secuencia mostrada en la figura se repite indefinidamente, ya que abarca todo un hiperperiodo). La razón está, como se indicó anteriormente, en la armonicidad de los periodos.

Tabla 3. Tabla de Utilización y Prioridad

Tarea	Ci	Pi	Utilización	Prioridad
A	2	5	40%	Alta
B	3	10	30%	Media
C	6	20	30%	Baja

El test basado en la utilización tiene pues la ventaja de ser muy sencillo de utilizar, pero tiene el inconveniente de no ser exacto, ya que no puede predecir que un sistema como el de la tabla es planificable.

7.9.6 Análisis Basado en el Tiempo de Respuesta. Los tests anteriores están basados en el concepto de utilización: en ambos se comprueba que la demanda de procesador por parte de las tareas no supera un determinado umbral. Otros estudios han trabajado en el terreno del tiempo, no en el de la utilización, con el objetivo de calcular cuánto tiempo le cuesta a una tarea *i* ejecutarse en el peor caso, teniendo en cuenta su propio tiempo de cómputo y el de otras tareas más prioritarias que puedan ejecutarse al mismo tiempo.

A este tiempo se le llama *tiempo de respuesta* de i, y se representa como *R*i. Una vez calculado *R*i, bastará con comprobar que el tiempo de respuesta se mantiene por debajo del plazo de la tarea, es decir, $Ri < D$i.

Al ejecutarse una tarea prioritaria, las demás tareas sufren la interferencia de una tarea de mayor prioridad, ya que éstas expulsan a las menos prioritarias para tomar el procesador cada vez que se activan. Así pues, para una tarea *i* se tiene:

$$Ri = Ci + Ii$$

Donde *Ii* es la *interferenci*a, o tiempo consumido por tareas de mayor prioridad durante la ejecución de i. La máxima interferencia sobre *i* se produce cuando ésta se activa al mismo

tiempo que todas las tareas más prioritarias que i, es decir, en su instante crítico. Existe una ecuación recurrente que permite calcular el peor tiempo de respuesta de una tarea.

$$Ri^{n+1} = Ci + \sum_{\forall j \mathrm{E} hp(i)} \left\lceil \frac{R_i^n}{P_j} \right\rceil C_j$$

Por ejemplo, si se tiene una tarea con un segundo orden de prioridad podemos decir otra tarea se ejecutó con anterioridad, la cual se debe tener en cuenta para el calculo del tiempo de respuesta de la tarea que se esta analizando.

Tabla 4. Tabla para Tiempo de Respuesta

Tarea	Pi(ms)	Ci(ms)	Di(ms)
sensor	4	1	4
monitoreo	1000	50	1000

Al Aplicar la ecuación recurrente para la tarea monitoreo tenemos:

$R^0 = 0$

$R^1 = 50$

$R^2 = 50 + \left\lceil \frac{50}{4} \right\rceil 1 = 62.5$

$R^3 = 50 + \left\lceil \frac{62.5}{4} \right\rceil 1 = 65.625$

$R^4 = 50 + \left\lceil \frac{65.625}{4} \right\rceil 1 = 66.406$

$R^5 = 50 + \left\lceil \frac{66.406}{4} \right\rceil 1 = 66.601$

$R^6 = 50 + \left\lceil \frac{66.601}{4} \right\rceil 1 = 66.650$

$$R^7 = 50 + \left\lceil \frac{66.650}{4} \right\rceil 1 = 66.662$$

$$R^8 = 50 + \left\lceil \frac{66.662}{4} \right\rceil 1 = 66.665$$

$$R^9 = 50 + \left\lceil \frac{66.665}{4} \right\rceil 1 = 66.666$$

$$R^{10} = 50 + \left\lceil \frac{66.666}{4} \right\rceil 1 = 66.666$$

Para finalizar el ejemplo podemos decir que el tiempo de respuesta se estabilizó en 66.666 ms lo cual es inferior al plazo de entrega que es de 1000 ms. Luego el sistema cumple con las características del Tiempo Real.

8. LENGUAJE ADA COMO LENGUAJE PARA TIEMPO REAL

Un lenguaje de programación es una herramienta utilizada por el programador con el propósito de expresar la solución a un problema en términos que el computador pueda entender (Smith,2001). Basándose en lo anterior podemos decir, que el programador recurrirá al uso del lenguaje de programación que mejor se ajuste a la solución ha desarrollar.

Si la intención del programador es desarrollar software que involucre Tiempo Real lo ideal es recurrir a un lenguaje de programación que de alguna manera permita crear soluciones eficientes y sobretodo garantice que éstas cumplan con lo que propone el Tiempo Real.
Dado lo anterior, se plantea el uso del lenguaje de programación Ada debido a las siguientes características:

- Confiabilidad: Ada es un lenguaje altamente confiable porque permite encontrar errores en tiempo de compilación, esto se debe en parte a que es fuertemente tipado, en contraposición a C o C++ los cuales muestran errores complejos en tiempo de ejecución.
- Abstracción: Ada es un lenguaje de alto nivel que incluye concurrencia y características de Tiempo Real que son innatas en el mismo. Esto hace que la implementación sea más confiable, ya que el compilador tiene la habilidad de hacer un sinnúmero de chequeos semánticos, esto le da ventaja sobre lenguajes secuenciales como C o C++.
- Predecible: Ada presenta mecanismos para el manejo de Tiempo Real como son el manejo de planificación por prioridades, implementación de las tareas, e intercambio y sincronización de las mismas en tiempo de ejecución según las prioridades haciendo de la aplicación en si misma muy robusta y predecible.
- Disponibilidad: Gnat, el compilador para ada, es de alta calidad, permitiendo un gran ambiente de desarrollo para Ada95, el Gnat incluye código nativo y un cross compilador para varias plataformas comunes, además la disponibilidad del compilador es amplia ya que es totalmente gratis.

- Penetración Industrial: Aunque C y C++ son más conocidos en ambientes industriales, Ada es la mejor opción pues esta hecho para implementar aplicación criticas y mas si son en Tiempo Real.
- Tipado fuerte: Ada es un lenguaje fuertemente tipado, lo que permite hacer aplicaciones robustas, además es un lenguaje que permite el uso de la programación orientada a objetos.
- Gnat: El compilador de gnat tiene una gran potencialidad frente a otros, porque usa el sistema gcc lo que permite compilar no solo en Windows o Linux sino también trabajar con fortran, C/C++ o Ada (Alonso, 2001).

Ada es un lenguaje de la programación de alto nivel pensado para las aplicaciones en vías de desarrollo donde la exactitud, seguridad, fiabilidad, y manutención son primeras metas (Burtch, 2001). Ada es un lenguaje fuerte del tipo orientado a Objetos, originalmente diseñado para aplicaciones militares, Ada es un lenguaje de propósito general que puede ser usado para cualquier problema. Tiene una estructura de bloque y un mecanismos de tipo de datos igual que Pascal, aunque con extensiones para aplicaciones de Tiempo Real y distribuidas. Provee una forma más segura de encapsulación que Pascal.

8.1 COMO NACIO ADA?

En los 70's hubo interés del Departamento de Defensa de EE.UU. para desarrollar un lenguaje sencillo para usar en sistemas de tiempo real incrustados. El Grupo de Trabajo de Lenguaje de Alto Orden (HOLWG) fue creado para diseñar este lenguaje. Este grupo revisó cerca de 500 lenguajes usados para desarrollar diversas aplicaciones militares. A través de una serie sucesiva de especificaciones recolectadas desde 1975 a 1978 se obtuvieron los requerimientos para tal lenguaje fueron definidos.

Se buscaron desarrolladores para este lenguaje y en pocos meses se obtuvieron 17 propuestas de las cuales 4 fueron elegidas. De las 4 finalistas nombre-código "red", "green", "yellow" y "blue" el lenguaje "green" del francés Jean Ichbiah fue elegido en 1979.

Aunque originalmente fue nombrado DoD-1, el nombre fue cambiado a ADA en honor a Ada Lovelace una pionera en computación y partidaria de Charles Babbage. Pascal fue el punto de partida para el diseño de ADA pero el lenguaje resultante es muy diferente en muchos aspectos. Ada es más extenso, más complejo, permite ejecución concurrente, control en Tiempo Real de la ejecución, manejo de excepciones y tipos de datos abstractos. El lenguaje fue estandarizado en 1983 tanto como estándar comercial de E.U.A., estándar militar de EE.UU. y estándar Internacional ANSI. La ultima revisión del lenguaje duro mas de 6 años desde 1988 hasta 1995 el proyecto llamado 9X donde la X fue cambiada por 5, hizo algunas correcciones al estándar de 1983 así como la implementación de mejores datos orientados a objetos, librerías jerárquicas y un mejor modelo de tareas para procesos. De esto sale la implementación de ADA 95, la cual es la versión actual de Ada.

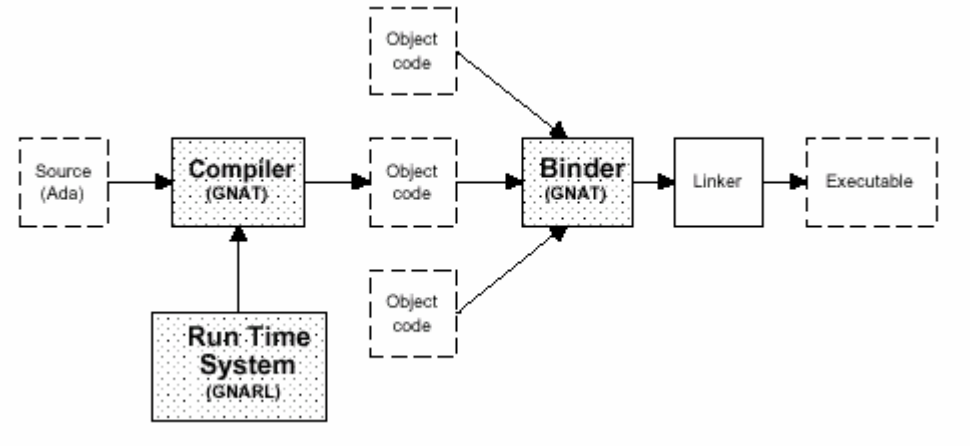
8.2 DESCRIPCIÓN DEL COMPILADOR

El compilador de Ada esta compuesto por dos partes, el front end y el back end. El front end esta formado por el GNAT y el back-end esta compuesto por el GCC, ambos el GNAT y el GCC se encargan de la compilación de los programas desarrollados en Ada., compilación que será explicada en detalle mas adelante.

El GNAT fue un producto elaborado por dos grupos interdisciplinarios que se dedicaron a este desarrollo. El primer grupo llamado *GNAT Development Team* formado en la Universidad de Nueva York y liderado por los profesores Edmond Schonberg y Robert B.K. Dewar, se encargaron del desarrollo del *front end* del compilador de Ada. El segundo grupo, llamado *Project PART Team* fue el encargado de desarrollar el *Ada Run-Time Library*. Este grupo hace parte de la Universidad de la Florida y fue liderado en sus propósitos por el profesor Theodore P. Baker.

En un principio este proyecto fue patrocinado por el gobierno de los Estados Unidos, hoy en día los principales creadores del proyecto forman parte de la compañía *Ada Core* la cual fundaron hace un tiempo, dicha compañía brinda soporte técnico bajo contrato a las entidades que utilicen el Gnat en el desarrollo de productos comerciales o industriales.El proceso de compilación esta formado por 3 partes principales, ellas son: el Compilador, el Run Time System y el Binder, las cuales podemos observar en la Figura 13.

Figura 13.Proceso de Compilación Gnat(Miranda,2002)

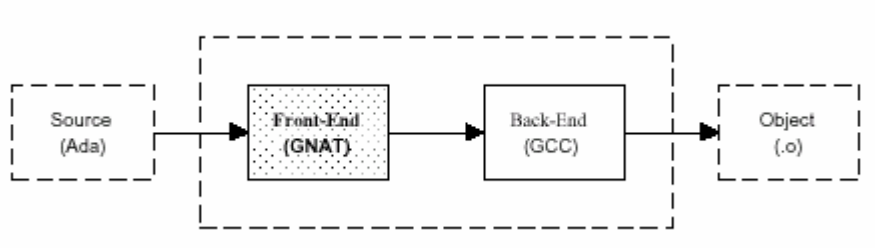


En la figura 13, se puede observar la estructura completa del Gnat, la cual consta de tres partes como se mencionó anteriormente. El código fuente en Ada es compilado y junto con el Run Time System, se generan 3 códigos objeto los cuales deben estar exentos de errores, posteriormente los objetos pasan a través de una etapa conocida como *Binder,* la cual se encarga de verificar la consistencia de los objetos y determina un orden de elaboración de los objetos que deben ser ensamblados en un archivo ejecutable.

El compilador esta compuesto por dos partes el *front end* (GNAT) y el *back end*(GCC). Tal como lo muestra la Figura 14 el resultado final de la compilación es un objeto (.o), el cual se forma como producto de las 5 fases que constituyen la compilación: *lexical analysis*, *syntactic analysis (parsing), semantic analysis*, *AST expansion*, y finalmente *AST*

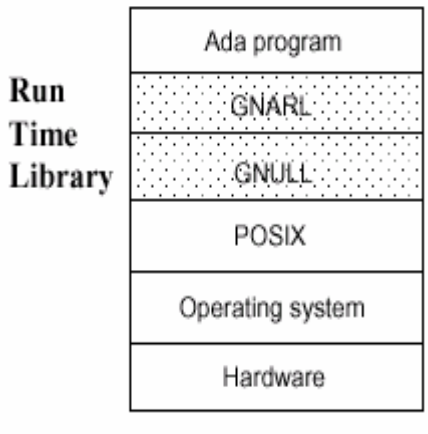
transformation(Miranda,2002), estas fases que conforman el proceso de la compilación no son otra cosa que etapas encargadas de la depuración del código y de la preparación del mismo para establecer un puente entre el *front end* y el *back end.*

Figura 14. Compilador Gnat(Miranda,2002)



En la Figura 14 vemos como el código fuente escrito en Ada se trasforma en un objeto de extensión .o después de haber pasado por la etapa de compilación compuesta por el GNAT y el GCC, los cuales componen el *front end* y el *back end* de la etapa de compilación.

Figura 15. Plataforma Gnat(Miranda,2002)



La figura 15 nos muestra un gráfico jerárquico donde podemos observar en la parte superior la aplicación desarrollada en Ada, la cual esta soportada sobre un plataforma que involucra la librería para Tiempo Real *Run Time Library* suministrada por el GNAT, dicha librería

esta compuesta por el GNARL (Gnu Ada Run Time Library), este componente es el encargado de recibir la solicitud de la aplicación para servicios que deban hacerse en Tiempo Real, como por ejemplo el manejo de la concurrencia entre tareas. Este tipo de solicitudes son atendidas por los componentes que contiene la capa POSIX. Mientras tanto, la librería GNULL se encarga de comunicar las dos capas del modelo jerárquico y además de eso le brinda portabilidad a la aplicación de acuerdo a las normas Posix para aplicaciones de Tiempo Real.

8.3 DESCRIPCION DEL LENGUAJE

Ada esta hecho para soportar la construcción de grandes programas. Un programa en Ada esta ordinariamente diseñado como una colección de grandes componentes de software llamados paquetes "Packages" cada uno representando un tipo de dato abstracto o un conjunto de objetos de datos compartidos entre subprogramas. Un programa en Ada consta de un procedimiento singular que sirve como programa principal, el cual declara variables, y ejecuta sentencias, incluyendo llamadas a otros subprogramas. Un programa en Ada puede envolver tareas que se ejecuten concurrentemente, si esto pasa entonces estas son inicializadas directamente por el programa principal y forman el nivel superior de la estructura del programa.

Ada provee un gran número de tipos de datos, incluyendo enteros, reales, enumeraciones, boleanos, arreglos, records, cadena de caracteres y apuntadores. Abstracción y encapsulación de tipos de datos y operaciones definidas por el usuario son proveídas por la característica de "package" o paquete. El control de secuencia dentro de un subprograma utiliza expresiones y estructuras de control similares a Pascal. La estructura de control de datos de Ada utiliza la organización de estructura de bloque estática como en Pascal y además el lenguaje prosee llamadas a referencias no estáticas. Desde que los programas pueden ejecutar tareas concurrentemente, estos pueden correr subprogramas independientemente uno del otro. Un área de almacenamiento de memoria para objetos de datos construidos es requerida.

Una de las razones por las cuales se promociona en este trabajo de grado la utilización de Ada como lenguaje de programación para implementar Sistemas de Tiempo Real, es por el bajo grado de dificultad que se presenta a la hora de desarrollar en el lenguaje de programación como tal, el diseño propuesto con Hrt-Hood.

En el tema "7.4 DESCRIPCION DE OBJETOS EN HRT-HOOD" se puede observar como los objetos básicos para el diseño de un Sistema de Tiempo Real son los objetos: Activos, Pasivos, Esporádicos, Protegidos y Periódicos, la combinación de cada uno de estos objetos va a permitir expresar de manera práctica la solución a una sistema que requiera Tiempo Real.

8.4 REPRESENTACION DE OBJETOS DE HRT-HOOD EN LENGUAJE ADA

Como se citó en párrafos anteriores Hrt-Hood aparte de ser una metodología orientada al diseño de objetos para Tiempo Real, tiene la particularidad de presentar cierto grado de facilidad a la hora de codificar en lenguaje Ada el diseño previo del sistema en Hrt-Hood, esto en cuanto a que lenguaje Ada presenta características que permiten al programador plasmar en código Ada los objetos del diseño junto con todas sus interacciones, como lo son: envió y recepción de datos, sincronización de objetos, y creación de mensajes de excepción.

Lenguaje Ada presenta ciertos elementos que van a contribuir en la implementación de sistemas de Tiempo Real. Dichos elementos se dividen en actividades, y mecanismos de coordinación o sincronización. Las actividades también conocidas como tareas se dividen en: Tareas Periódicas que corresponde a los objetos ciclos en Hrt-Hood, tareas aperiódicas u objetos esporádicos para Hrt-Hood, tareas protegidas u objetos protegidos en Hrt-Hood, las tareas en lenguaje Ada no requieren de mensajes externos para arrancar su proceso.

8.4.1 Tareas Periódicas. Las tareas periódicas al igual que los objetos cíclicos corresponden a actividades realizadas de manera repetitiva, y son muy útiles en algoritmos de control o monitoreo de procesos, debido a que pueden ser implementadas para sensar el estado de variables.

La declaración o especificación de una tarea periódica se hace de la siguiente manera:

```
task Periódica; -- Declaración de la tarea periódica
```

El cuerpo de la tarea se codifica en Ada de la siguiente de la siguiente forma:

```
task body periodica is –Declaracion del cuerpo de la tarea
        periodo : constant; duracion := p; --Declaración del periodo(Pi) de la tarea.
        siguiente : time;
  begin
        -- acciones iniciales;
        -- sincronización
        siguiente := clock;-- La variable siguiente toma el valor del esto del reloj
        loop --Empieza el ciclo
          -- acciones de la tarea;
          siguiente:=siguiente+periodo; --La variable siguiente es actualizada
         delay until siguiente; --Esta acción impone un retardo hasta un nuevo periodo
        end loop—Finaliza el ciclo
end periodica; --Final de la tarea
```

8.4.2 Tareas Aperiódicas. Las tareas Aperiódicas representan en Hrt-Hood los objetos esporádicos, a diferencia de las tareas periódicas, las tareas aperiódicas esperan por la ejecución de un evento para realizar su proceso.

La especificación de una tarea aperiódica se realiza de la siguiente manera:

```
task Aperiodica is;--Declaración de la tarea Aperiódica.
  entry nombre_evento(Declaración de variables);
end Aperiodica;--Final de la declaración o especificación.
```

El cuerpo de una tarea aperiódica se codifica de la siguiente forma:

```
task body Aperiodica is – Cuerpo de la tarea
 begin
        -- Acciones iniciales
        loop
     select
      accept evento;
      -- Acciones de la tarea
      or
      delay max;
      -- excepción
    end loop;
end Aperiodica; -- Final de la tarea
```

8.4.3 Objetos Protegidos. Los objetos protegidos o actividades protegidas representan aquellos procesos que requieren de todo el cuidado de la unidad de procesamiento para evitar que los datos sean dañados, por ejemplo en el proceso de escritura y lectora de un buffer compartido se debe proteger el mismo, para que los procesos de escritura y lectura de datos no se realicen al mismo tiempo sobre el buffer, evitando así la corrupción de los datos.

La declaración de un objeto protegido es la siguiente:

```
protected  type nombre_objeto_protegido  is --Declaración de la tarea como protegida
   procedure  procedimiento_inicial( Decla.variables); --Entrada de dato al Objeto
   procedure procemiento_final( Delca. Variables ); --Salida de dato del Objeto
private
   --Declaración de las variables a usar en la tarea
end nombre_objeto_protegido;
```

El cuerpo de un objeto protegido se codifica de la siguiente forma:

```
protected  body nombre_objeto_protegido is – Cuerpo del objeto protegido
   procedure procedimiento_inicial(Decla.variables)  is –Entra  el dato al Objeto
      begin
            -- Porceso del objeto
  end procedimiento_inicial;
 procedure procemiento_final( Delca. Variables ) is Se retorna el resultado
      begin
        --Sale el resultado del objeto
  end procemiento_final;
end nombre_objeto_protegido;
```

8.4.4 Objetos Pasivos y Objetos Activos. La codificación en Ada de los objetos pasivos y activos no requiere de consideraciones especiales más allá de aquellas que nos expresan sus conceptos. En el caso de los objetos pasivos podemos decir que su utilización se realiza para almacenar variables y ejecutar funciones y procedimientos que tienen libre acceso por los procesos y que generalmente no representan información crítica. Los objetos pasivos pueden ser representados en lenguaje Ada por funciones y procedimientos que no necesariamente involucren Tiempo Real. Por otro lado, los objetos activos son objetos que tienen actividad propia y que además tienen la particularidad de incluir o usar todo tipo de objetos entre los cuales se pueden contar aquellos que utilizan Tiempo Real en sus procesos. Un objeto activo puede ser representado en lenguaje Ada por funciones que expresen cualquier tipo de actividad o por procedimientos que involucren Tiempo Real por medio de tareas aperiódicas o periódicas.

8.5 MECANISMOS DE SINCRONIZACIÓN

Los mecanismos de sincronización también se conocen como mecanismos de coordinación, dichos mecanismo cumplen una función específica y es establecer un enlace entre ciertos objetos del sistema, en este caso las tareas. Cuando un sistema en Tiempo Real se compone de varias tareas, estas de alguna manera debe tener cierta coordinación a la hora de interactuar entre si, esto debido a que en Ada las tareas empiezan a ejecutarse sin ningún control y pueden presentarse incoherencias en el momento en que las mismas tareas deben comunicarse entre ellas o cuando se requiera que estas lancen datos referentes a resultados de un proceso. Entre las tareas los mecanismos de sincronización mas conocidos son las señales y las barreras.

8.5.1 Señales. Una señal no es otra cosa que un mecanismo de sincronización que permite crear una condición de espera en una tarea mientras aguarda por un indicador que le permita continuar con su proceso.
Por ejemplo si tenemos dos tareas periódicas llamadas tarea_1 y tarea2_ las cuales captura los datos de dos sensores llamados sensor_1 y sensor_2 respectivamente, podemos tener conflictos cuando dichas tareas accedan a un solo conversor análogo digital con que cuenta el sistema, la solución a este conflicto es que una tarea sea mas prioritaria que la otra y cada vez que ésta termine el uso del conversor envié una señal a la otra para que pueda usar el conversor sin problema, de esta manera se evita que ambas tareas accedan al conversor en el mismo instante de tiempo.

Dos acciones condicionan el uso de una señal, la acción *esperar* con un guarda asociado (la variable *llego*), la cual permite bloquear una tarea hasta que una señal llegue. La acción *enviar*, que cambia el valor de la variable *llego*, permitiendo avisar a la tarea bloqueada, que la señal ha llegado.

```
package señales is -- Declaración del paquete señal
  protected type señal is --Declaración del objeto protegido señal
    procedure enviar;
    entry esperar;
  private
    llego : Boolean:= False;
  end señal;
end señales;

package body señales is
  protected body señal is
      procedure enviar is
       begin
   llego:= true;
end enviar;

entry esperar when llego is
   begin
     llego:= false;
   end esperar;
 end señal;
end señales;
```

8.5.2 Barreras. Debemos entender este mecanismo de sincronización como el partidor de una competencia hípica, debido a que las tareas experimentan un bloqueo y no son liberadas hasta que la totalidad de las mismas este en igualdad de condiciones para ser ejecutadas, es decir, lo que busca una barrera es imponer una condición que establezca que todas las tareas empiezan a ejecutarse en el mismo instante, lo que permite sincronizar el inicio de ejecución de las mismas.

Entre las consideraciones se tiene: en el código Ada, se utiliza una variable de tipo *Boolean* (liberar), con el objetivo de permitir mantener la barrera levantada o abierta una vez se ha completado el número de tareas encoladas. La variable *total*, contiene el número de tareas que serán encoladas a la espera de que hayan llegado todas, éste parámetro debe ser inicializado en el momento de la instanciación.

```
package Barreras is
   protected type Barrera is
     entry esperar;
   private
    liberar: Boolean:=False;
    todas: Integer:= ...;
   end Barrera;
end Barreras;

package body Barreras is
  protected body barrera is
   entry esperar when esperar´count = total or liberar is
   begin
    if esperar´count > 0 then
       liberar := true;
    else
       liberar := False;
    end if;
   end esperar;
 end Barrera;
end Barreras;
```

8.5.3 Citas. La cita es un mecanismo que permite a dos tareas comunicarse sincrónicamente y bidereccionalmente. Su utilización es bastante común en la comunicación sincrónica de tareas y de uso en todo tipo de aplicaciones industriales.

```
task type nombre_de _tarea is       --Declaración de la tarea
  entry  nombre_de_la_cita( Decalacion de variables ); --Rendezvous(cita)
end nombre_de _tarea;

task body nombre_de _tarea is
begin
  accept nombre_de_la_cita (Declaración de variables) do --Sincronización
      --Proceso que realiza la tarea
  end nombre_de_la_cita;
end nombre_de _tarea;
```

9. EJEMPLO DE CONTROL Y MONITOREO DE UN TANQUE EN TIEMPO REAL

Como ejemplo de aplicación de este trabajo de grado**,** de titulo ***flujo de diseño para el desarrollo de aplicaciones en Tiempo Real usando lenguaje ada*** se ha tomado el proyecto que se ha venido desarrollando con el grupo de investigación *Desarrollo Hardware y Software para Procesos de Ingeniería*, de la *Corporación Universitaria Autónoma de Occidente* (Dirigido por el docente Diego Martínez y del cual formo parte).

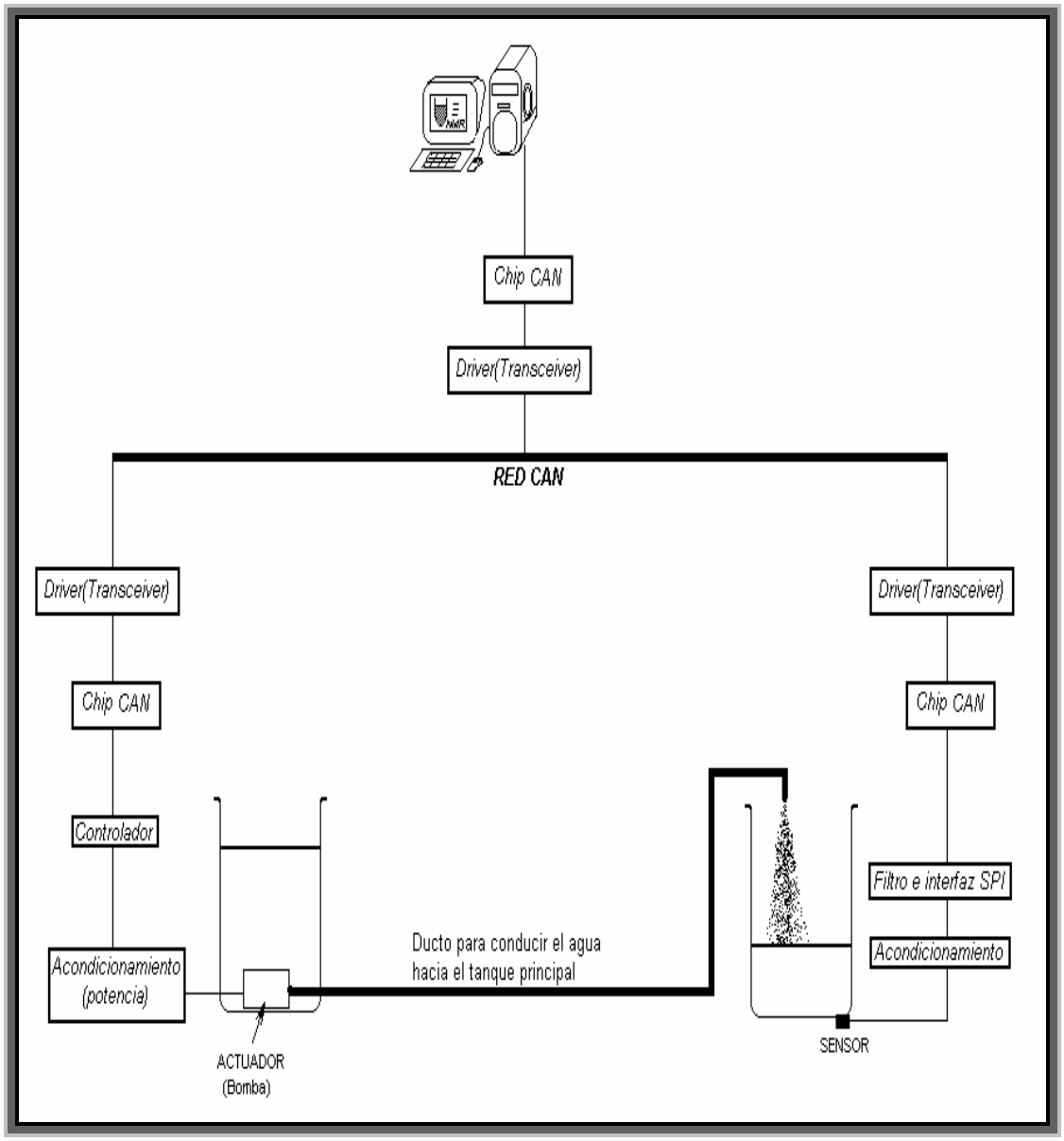
El proyecto se basa en el monitoreo y control de un tanque en Tiempo Real, para lo cual se ha desarrollado una plataforma que consta de varios niveles, un nivel inferior del cual hacen parte el tanque a controlar y el tanque de llenado, este nivel también lo constituyen el actuador y el sensor de los tanques. El segundo nivel o nivel medio esta compuesto por la red que comunica el nivel inferior y el nivel superior, el cual trataremos mas adelante. El nivel medio esta formado por la red implementada con el protocolo de bus de campo CAN, dicho protocolo soporta Tiempo Real he ahí la razón de su uso. Por ultimo tenemos el tercer nivel o nivel superior, este nivel cuenta con una computadora donde se ejecuta un programa desarrollado en Ada y que funciona en Tiempo Real, este programa se encarga de comunicarse con la red CAN con el propósito de obtener los datos necesarios para monitorear el nivel del tanque y tener la capacidad de modificar el set point del sistema, todo esto lo podemos ver con mas detalle en la figura 16.

Dada la complejidad del proyecto, se tomó la decisión de dividir el mismo en varios proyectos de grado entre los cuales se encuentra el proyecto de grado que se esta tratando además de otros dos proyectos: MODELAMIENTO FORMAL DE SISTEMAS CONTINUOS DE CONTROL INDUSTRIAL EN AMBIENTES DISTRIBUIDOS, este proyecto de grado fue desarrollado por Norberto Muñoz y ofrece una alternativa de diseño formal de sistemas de control industrial usando redes de petri. Otro proyecto de grado que se desprende del proyecto original lleva como titulo IMPLEMENTACIÓN DE UN MÓDULO DE SENSADO Y DE ACTUACIÓN PARA EL CONTROL DE NIVEL DE

UN TANQUE MEDIANTE LA TÉCNICA DE CODISEÑO HARDWARE-SOFTWARE, este proyecto fue adelantado por Andrés Macias y Jorge Lizcano y comprende el desarrollo del sistema de control de los tanques usando la técnica de codiseño Hardware-Software.

Cabe anotar que en nuestro trabajo hemos tenido un gran apoyo por parte de Eduardo Velásquez quien adelanta un proyecto de grado llamado DISEÑO E IMPLEMENTACION DE UNA RED PARA CONTROL Y MONITOREO REMOTO DE UNA PLANTA DE NIVEL y nos ha compartido sus conocimientos en cuanto el manejo del protocolo CAN y el desarrollo de la tarjeta ISA que requiere el computador para comunicarse con la red CAN.

Figura 16. Esquema global del sistema de tanques

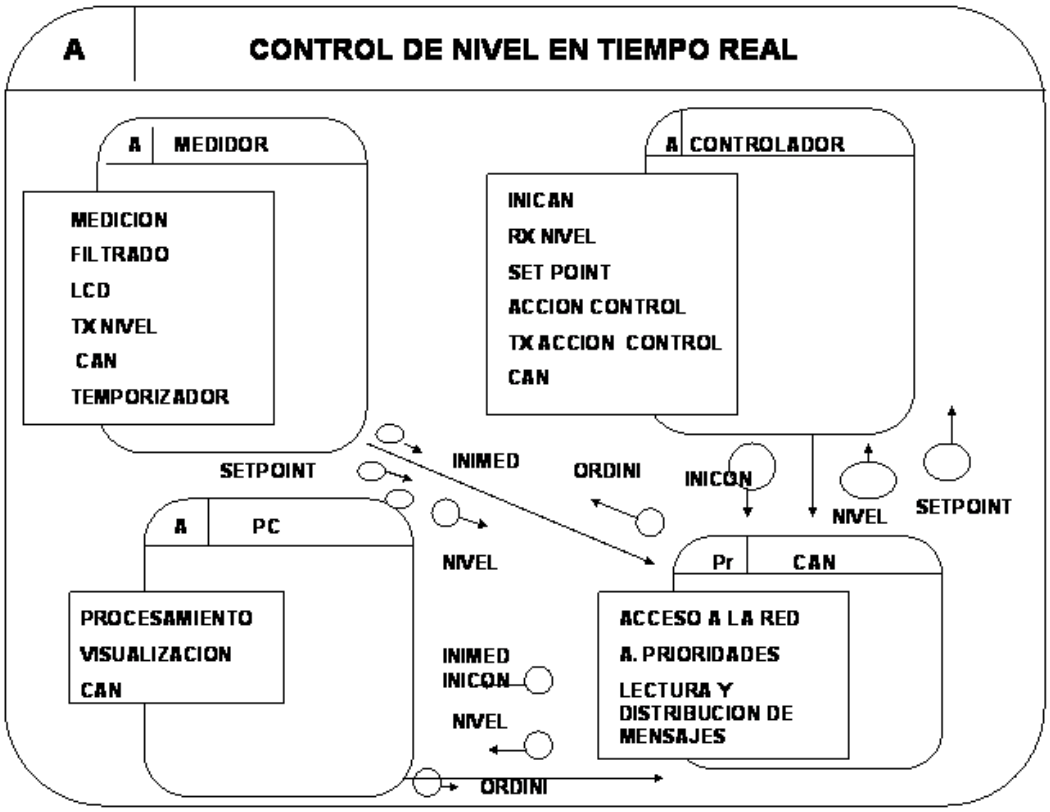


9.1 DISEÑO DEL SISTEMA EN HRT-HOOD

Tal como se comentó en temas anteriores, uno de los pilares propuestos para diseñar Sistemas en Tiempo Real es el uso de Hrt-Hood. Esta herramienta nos ayuda a definir los requisitos de nuestro diseño de manera concreta, además permitirnos la creación de los componentes de nuestra arquitectura lógica.

El primer modelo de nuestro diseño de Hrt-Hood comprende un gran objeto padre llamado *CONTROL DE NIVEL EN TIEMPO REAL*, este objeto a sido designado como activo, entre otras cosas por contener dentro de si actividad propia la cual es representada por sus objetos hijos, llamados *Medidor(A*ctivo*), Controlador(*Activo*), Pc(*Activo*), Can(Protegido),* ver figura 17..

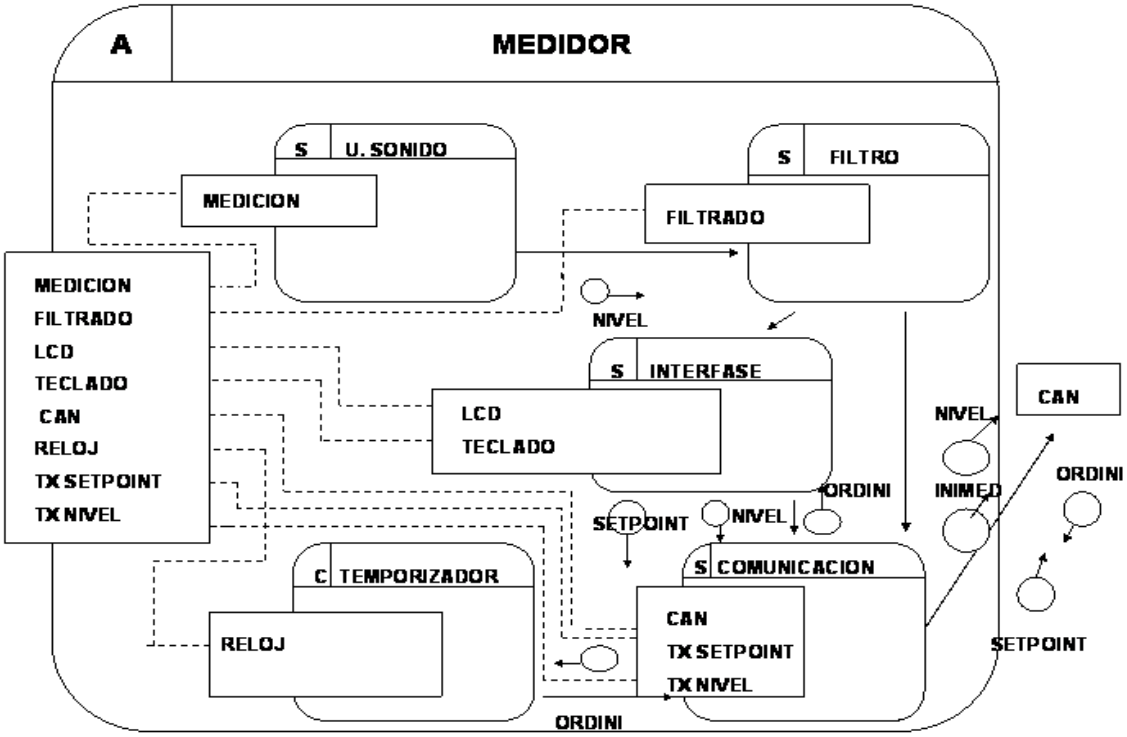
Figura 17. Modelo en HRT-HOOD: primer nivel de jerarquía del sistema



- Objeto *medidor*: Su función es la de tomar periódicamente una medición de nivel en el tanque a controlar, y acondicionarla de tal forma que tenga un formato válido para ser llevada al objeto CAN. El objeto medidor se clasificó como activo porque en su interior contiene elementos que tienen actividad periódica.
- Objeto *controlador*: Recibe un valor de nivel actual, desde la Interfaz CAN y con base en él calcula una acción de control y de esta forma controlar la cantidad de líquido que entra al tanque.
- Objeto *PC*: A este sitio llega información del estado del actuador, del nivel actual del tanque e información de sincronización del sistema, todo ello con el fin de tener reportes del comportamiento del sistema en general, o para realizar monitoreo remoto, de la planta en un computador.
- Objeto *CAN*: Representa las funciones del protocolo CAN y el medio de transmisión para que la información fluya desde un terminal hacia otro. Recibe información de nivel y de setpoint (SP_Teclado) desde el *medidor*, para enviársela al controlador y al PC, recibe información del estado del actuador desde el controlador, información que posteriormente será enviada al objeto PC.

Los objetos anteriores han sido designados como objetos activos, dado que en cierta forma representan actividad propia, característica innata de objetos que deben ser designados como activos. El objeto CAN a pesar de tener características de objeto activo como lo es el hecho de presentar actividad propia, ha sido designado como un objeto protegido debido a que este objeto debe tener un control estricto del acceso a la red, puesto que los demás objetos no pueden leer y escribir sobre la misma en igual instante de tiempo.

Figura 18. Modelo en HRT-HOOD: segundo nivel de jerarquía objeto *medidor*



El objeto activo medidor tiene en su interior varios objetos, los cuales se consideran objetos hijos, estos objetos se encargan de realizar las operaciones del objeto *Medidor*. Entre los objetos hijos del objeto *Medidor* tenemos: El objeto *Ultrasonido*, el objeto *Filtro*, el objeto *Temporizador*, el objeto *Interfase* y el objeto *Comunicación*. El objeto *Ultrasonido* es un objeto esporádico encargado de la medición, el Objeto *Filtro*, se encarga de darle una mejor definición a las señales de nivel que son enviadas al objeto *Comunicación* para ser transmitidas por este mismo objeto a Can. El objeto *Interfase* recibe los datos que son introducidos por teclado y también se encarga de mostrar en una pantalla de LCD, los datos de Setpoint y Nivel. También podemos comentar que en el objeto *Temporizador* recibe la orden de inicio (ORDINI) del objeto *Pc* por medio de los objetos *Can* y *Comunicación.*

El objeto activo *Controlador* (Figura 19) contiene dos objetos que realizan actividades esporádicas uno es el objeto *Comunicación* y el otro es el objeto *Control.* El primero se encarga de interactuar con el objeto *Can* para enviar y recibir flujo de datos a la red Can, el objeto *Comunicación* envía la acción de control (A.CONTROL), el dato de inicialización (INICON), y recibe el Set Point o referencia , y el Nivel, mientras que el objeto Controlador se encarga de generar la acción de control de acuerdo a los datos de Nivel y Set Point que recibe del objeto *Comunicación*; éste a su vez también recibe la acción de control para luego enviarla al objeto *Can*, como se dijo anteriormente.

Figura 19. Modelo en HRT-HOOD: segundo nivel de jerarquía objeto *controlador*

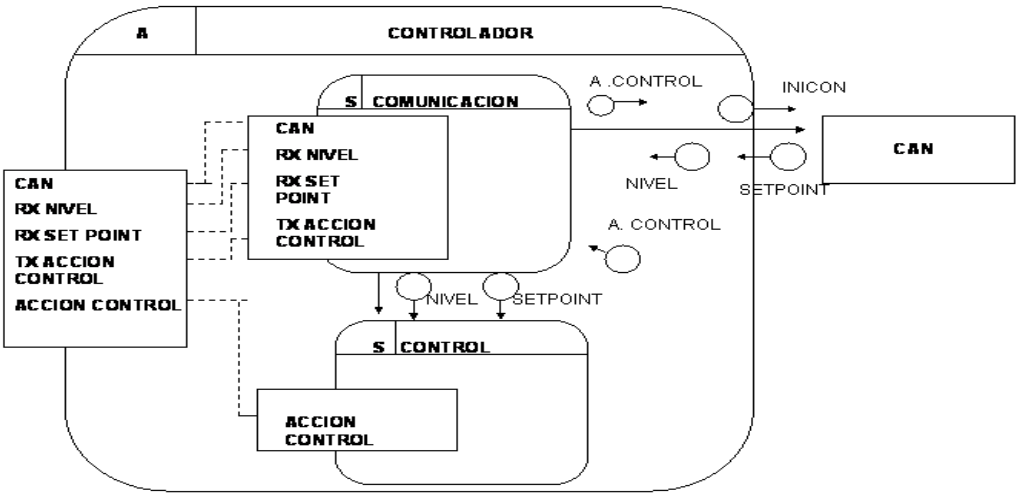
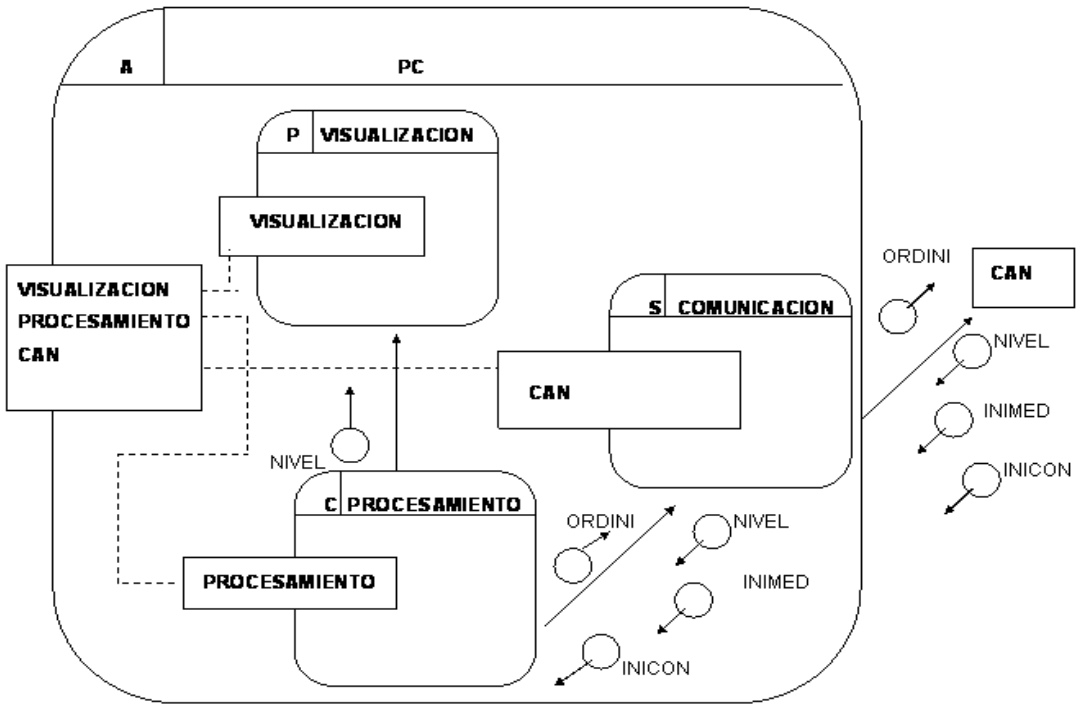


Figura 20. Modelo en HRT-HOOD: segundo nivel de jerarquía objeto *PC*



El objeto activo *PC*, representa la recepción y la visualización en Tiempo Real de los datos que son enviados a la computadora por el objeto *Can* .

El objeto *Comunicación*, objeto hijo de *Pc,* recibe por medio del objeto *Can* los datos de Nivel, inicio del objeto Medidor(INIMED), inicio del objeto controlador(INICON) , con el fin de ser enviados a el objeto cíclico Procesamiento , que se encarga de procesarlos para luego enviar el dato de Nivel en Tiempo Real a el objeto *Visualización* . En el objeto procesamiento también se produce la orden de inicio del programa (ORDINI), esta orden es enviada a el objeto *Can* para que los demás objetos se sincronicen junto con el objeto *PC* .

9.2 PLANIFICACION DEL SISTEMA

Dado que la aplicación realizada en lenguaje Ada tiene como principal objetivo tomar los datos del estado de nivel del sistema para ser visualizados en Tiempo Real, se desarrolló una única tarea llama Procesamiento la cual cumple con el propósito deseado.

Para la evaluación del tiempo de respuesta del sistema, se utilizo la medición del tiempo de computo *(Ci*) de la tarea para lo cual se utilizó un Osciloscopio digital. En esta prueba del sistema obtuvimos un tiempo de computo de 3.8 ms, para un periodo de 0.250 segundos y un plazo de igual valor.
Dado que no existen mas hilos de control dentro de la computadora, el análisis de planificabilidad se reduce a verificar la siguiente desigualdad:

$$Ci < Di \Rightarrow 0.0038seg < 0.250seg$$

Como se puede apreciar, la misma se satisface, indicando que el sistema es planificable.

Se realizaron las mismas pruebas tanto en RTLinux con RTLGnat como en Windows 98 SE, donde se pudo destacar como el tiempo de computo en RTLinux (3.8ms) es mucho menor que en Windows (5ms), además en RTlinux nunca varió el tiempo de computo de la tarea cuando se ejecutaron otras aplicaciones del sistema operativo, mientras que en Windows el mismo fue inestable.

10 CONCLUSIONES

Dado que en Colombia no solo son escasos los grupos de investigación que se enfocan en el desarrollo de Sistemas de Tiempo Real sino que también son pocos los estudios sobre el mismo tema, se desarrolló un trabajo de grado que plantea un flujo de diseño para el desarrollo de aplicaciones de control de procesos continuos que requieren Tiempo Real, utilizando Ada como lenguaje de síntesis, con el propósito no solo de estudiar en profundidad el tema Tiempo real, sino también de permitirle a la academia contar con una metodología para el diseño e implementación de sistemas de este tipo, y que fuese acorde con las características y complejidades del mismo. Al finalizar dicho trabajo de grado se concluyo que:

- A través del análisis del comportamiento de los Sistemas de Tiempo Real y la investigación de métodos y lenguajes se comprueba que la metodología Hrt-Hood resulta la mas idónea para el diseño de este tipo de sistemas porque no solo permite ahorrar tiempo en horas de diseño debido a su fácil uso y comprensión sino que también presenta la ventaja de combinar el uso de componentes tanto de Software como de Hardware, dándole un carácter mas general a los sistemas desarrollados, lo que puede facilitar su adopción por parte de la industria colombiana.

- El lenguaje de programación Ada resultó ser un lenguaje ideal para el desarrollo de este tipo de aplicaciones, dado que presenta características acordes al concepto Tiempo Real, lo que lo convierten en un lenguaje predecible, además facilita el traslado de diseños de HRT-HOOD al código nativo del lenguaje. Por otro lado, Ada brinda confiabilidad y las bondades de los lenguajes orientados a objetos, además de ser un lenguaje fuertemente tipado.

- RTlinux es un núcleo ideal para montar aplicaciones para Tiempo Real desarrolladas en lenguaje ada, debido a su alta estabilidad y confiabilidad en la ejecución de tareas. Además,

soporta un planificador de prioridades estáticas, cuyas características son muy adecuadas para el desarrollo de aplicaciones críticas.

▪ RTLGnat resulto ser una aplicación confiable para portar programas en código Ada sobre el RTlinux, lo cual brinda la posibilidad de desarrollar aplicaciones que requieran Tiempo Real suave o duro.

▪ El proceso de desarrollo Iterativo presenta una estrecha relación con la metodología de diseño Hrt-Hood, al permitir plasmar los detalles abstractos de los sistemas en una serie de componentes formados a partir de objetos característicos de los sistemas de Tiempo Real. Tales como: objetos PASIVOS, ACTIVOS, PROTEGIDOS, CICLICOS y ESPORADICOS.

▪ Al tener acceso a una metodología de diseño como la promovida en este trabajo de grado se puede contar con soluciones flexibles, y fáciles de implementar dada su baja complejidad en la validación de los Sistemas de Tiempo Real, debido a que carece de cálculos complejos, haciendo atractiva dicha propuesta para la industria en general. De esta manera, se ratifica la eficiencia de la misma, puesto que por un lado representa menos tiempo de diseño de lo que nos tardaríamos con las metodologías tradicionales, ya que se requiere de menos introducción de líneas de código con llamados al sistema operativo y por el otro, reduce costos al utilizar menos horas de programación. Otra ventaja es que las herramientas usadas en esta tesis tales como: el compilador de Ada(el Gnat), el RTLGnat 1.0 y linux hacen parte del mundo del software libre lo que representa una gran disminución en el dinero que se debe invertir a la hora de desarrollar un Sistema que requiera Tiempo Real.

BIBLIOGRAFIA

ALONSO, Alejandro. y DE LA PUENTE, Juan. Using Linux and Ada in the Development of Distributed Computer Control Systems.[en línea], Madrid : Universidad Politécnica de Madrid, Departamento de Ingeniería de Sistemas Telemáticos,2001.[Citado :12 /6/2003]. Disponible por Internet : http://polaris.dit.upm.es/~jpuente/str/publications.html

BARBAZAN, Ángel. Sistemas Operativos En Tiempo Real. [en línea], Madrid :RTOS, 2000.[Citado : 07/8/2003].Disponible por Internet : http://www.disa.bi.ehu.es/spanish/asignaturas/ii/trabajos/RTOS.pdf

BALBASTRE, Patricia. Modelo De Tareas Para La Integración Del Control Y La Planificación En Sistemas De Tiempo Real. Valencia, 2002, 165p. Tesis Doctoral (Ingeniero Electrónico).Universidad Politécnica de Valencia. Departamento de Informática de Sistemas y Computadores.

BURTCH, Ken.The Big Online Book of Linux Ada Programming. [en línea], Canadá : Pegasoft, 2001. [Citado : 07/8/2003].Disponible por Internet : http://www.vaxxine.com/pegasoft/homes/book.html

BURNS, Allan. y Welllings, Andy. A Structured Design Method for Hard Real Time Ada Systems.York : Elsevier, 1995. 313 p

CORVALAN, Julio. Sistemas de Tiempo Real. [en línea], México : Grupo de Investigaciones y Desarrollos en el área de los Sistemas de Tiempo Real ,2003. [Citado : 20/9/2003].Disponible por Internet : http://www.led.uc.edu.py/str/revision.htm

DE LA PUENTE, Juan. Diseño de Sistemas de Tiempo Real.[en línea], Madrid : Universidad Politécnica de Madrid, Departamento de Ingeniería de Sistemas Telemáticos, 2001. [Citado :12 /6/2003]. Disponible por Internet : http://polaris.dit.upm.es/~jpuente/strl/transparencias/Desarrollo.pdf

GONZALEZ, Apolinar. Especificación De Componentes Mediante Redes De Petri Para El Diseño y Validación De Sistemas De Control De Tiempo Real. Valencia, 1999, 183 p. Tesis Doctoral (Ingeniero Electrónico). Universidad Politécnica de Valencia. Departamento de Informática de Sistemas y Computadores .

BARBOSA, Karina. Y ORTEGÓN, Ricardo. Concepto y Definición de Sistemas Operativos.[en línea], Veracruz : Instituto Tecnológico de Veracruz, 2003. [Citado :16 /10/2003]. Disponible por Internet : http://www.itver.edu.mx/so1/concepto.html .

MARTINEZ, Diego. Notas de Clase de Sistemas en Tiempo Real en Microprocesadores II. Corporación Autónoma de Occidente, 2002.

MARTÍN, James. Diseño de Sistemas de Computadores en Tiempo Real. México : Diana, 1980. 706 p.

MIRANDA, Javier. A Detailed Description of the GNU Ada Run Time. [en línea], Islas Canarias : Universidad de las Palmas de Gran Canaria, 2002. [Citado :18 /10/2003]. Disponible por Internet : http://www.iuma.ulpgc.es/users/jmiranda/

RIPOLL, Ismael. Tutorial de Real Time Linux. [en línea], Valencia : Disca, 2000. [Citado :18 /10/2003]. Disponible por Internet : http://bernia.disca.upv.es/rtportal/tutorial/rtlinux-tutorial.pdf

SMITH, Michael. Object-oriented Software in Ada 95. [en línea], Brighton: Universidad de Brighton,2001. [Citado :20 /10/2003]. Disponible por Internet : http://www.adaic.org/free/freebook.html

ZHANG, Wei. Stability analysis of networked control systems. Hong Kong : Universidad de Hong Kong, 2001. 83 p

ANEXO A(Ejemplo de una Tarea Periódica y una Tarea Aperiódica sincronizadas por una barrera en lenguaje Ada)

Este sencillo ejemplo nos demuestra como sincronizar dos tareas por medio de una barrera, en la tarea periódica llamada “tarea_periodica” se carga una variable llamada valor con el número 5 y luego este valor es enviado a la tarea aperiódica la cual lo acepta y los multiplica por 4 imprimiendo en pantalla el valor de 20. Lo destacable de este ejemplo es la forma como ambas tareas se sincronizan por medio de la barrera.

```
with ada.text_io,Ada.Integer_Text_Io;
use ada.text_io,Ada.Integer_Text_Io;
with Ada.Real_Time;
use Ada.Real_Time;

procedure barreras is

task tarea_periodica is --Declaración de la tarea periódica
   pragma priority(15);--Declaración de prioridad
end tarea_periodica;

task tarea_aperiodica is  --Declaración de la tarea aperiódica
   pragma priority(10); --Declaración de prioridad de la tarea aperiódica
   entry enviar1(Data: in integer);
end tarea_aperiodica;

protected type barrera is   --Declaración de la barrera
   entry esperar;
private
   liberar: Boolean := FALSE;
end barrera;

protected body barrera is  --Cuerpo de la Barrera
   entry esperar when esperar'Count=2 or
   liberar is
```

```
      begin
            if esperar'Count > 0   then
               liberar :=TRUE;
            else
               liberar :=FALSE;
            end if;
   end esperar;
end barrera;
thread : barrera;-- Declaración de una variable del tipo barrera

task body tarea_periodica is  -- Cuerpo de la tarea periódica
  periodo: time_span := milliseconds(500);
   siguiente : Time;
   valor: integer;
   begin
      thread.esperar;
      siguiente :=Clock;
         loop
             put("Tomo el valor de 5 ");
             New_line;
             valor:=5;
             tarea_aperiodica.enviar1(valor);
          siguiente:=siguiente+periodo;
        delay until siguiente;
      end loop;
end tarea_periodica;

task body tarea_aperiodica is  --Cuerpo de la tarea aperiódica
  D : integer;
  Answer: integer;
    begin
    thread.esperar;
      loop
          accept enviar1(Data : in integer) do
          D:=Data;
          answer:= D*4;    --acciones tarea
```

```
          put("el resultado es:");
          New_line;
          put(answer);
          New_line;
          end enviar1;
   end loop;
end tarea_aperiodica;
begin
null;
end barreras;
```

ANEXO B(Ejemplo de una Tarea Periódica y una Tarea Aperiódica sincronizadas por una señal en lenguaje Ada)

Este ejemplo es idéntico al ejemplo anterior solo que las tareas se sincronizan por medio de una señal y no por una barrera.

```
with ada.text_io,Ada.Integer_Text_Io;
use ada.text_io,Ada.Integer_Text_Io;
with Ada.Real_Time;
use Ada.Real_Time;

procedure señales is

task tarea_periodica is
   pragma priority(15);
end tarea_periodica;

task tarea_aperiodica is
   pragma priority(10);
   entry enviar1(Data: in integer);
end tarea_aperiodica;

protected type señal is
  procedure enviar;
  entry esperar;
private llego: boolean:=FALSE;
end señal;

protected body señal is
  procedure enviar is
    begin
      llego:=TRUE;
   end enviar;
   entry esperar when llego is
    begin
```

```
        llego:=false;
    end esperar;
end señal;

thread2 : señal;

task body tarea_periodica is
  periodo: time_span := milliseconds(500);
   siguiente : Time;
   valor: integer;
   begin
      thread2.enviar;
      siguiente :=Clock;
         loop
             put("Tomo el valor de 5 ");
             New_line;
             valor:=5;
             tarea_aperiodica.enviar1(valor);
          siguiente:=siguiente+periodo;
        delay until siguiente;
      end loop;
end tarea_periodica;

task body tarea_aperiodica is
  D : integer;
  Answer: integer;
    begin
    thread2.esperar;
      loop
          accept enviar1(Data : in integer) do
          D:=Data;
          answer:= D*4;    --acciones tarea
          put("el resultado es:");
          New_line;
          put(answer);
          New_line;
```

```
          end enviar1;
   end loop;
end tarea_aperiodica;
begin
null;
end  señales;
```

ANEXO C(Ejemplo de un Objeto Protegido en Ada)

Este ejemplo documenta como se trabaja con datos protegidos en lenguaje Ada, en el programa podemos observar como se declaran las tarea lectura_datos y proceso_datos como protegida y como se declaran sus componentes, ambas tareas reciben datos por medio de variables locales que son declaradas en el cuerpo del programa y luego estos datos son procesados en los cuerpos de las tareas protegidas y sus resultados impresos en pantalla

```
with ada.text_io,Ada.Integer_Text_Io;
use ada.text_io,Ada.Integer_Text_Io;
--with Ada.calendar;
--use Ada.calendar;

procedure protegido is
-------------------------- Declaración tareas protegidas-----------------
protected  type lectura_datos is --Declaración de la tarea como protegida
   procedure Start( F:in Positive ); --Proceso de entrada de dato a la
tarea
   procedure Finish( Result:out Positive ); --Proceso de salida de dato
private
 aux     : integer;--Declaración de las variables a usar en la tarea
 answer : Positive;
end lectura_datos;
-------------------------------------------------------------------------
protected type proceso_datos is-- Declaración segunda tarea
   procedure start(f:in positive);
   procedure finish(Result: out positive);
private
 aux2 : positive;
 answer2  : positive;
end proceso_datos;
--------------------------------Variables Globales----------------------
Thread_1 : lectura_datos;
Thread_2 : proceso_datos;
trabajo  : positive;
```

```
trabajo2 : positive;
----------------------------------- Cuerpos de las Tareas --------------

protected  body lectura_datos is
      procedure Start( F:in Positive ) is --  when 2<3 is--Se recibe el
dato de la cita
         begin
               aux:=f;
               answer:= aux*2 ;          --Se procesa el dato
       end start;
      procedure Finish( Result:out Positive ) is -- when 2<3 is --Se
retorna el resultado
         begin
              Result := Answer;
      end finish;
    end lectura_datos;

protected  body proceso_datos is
    procedure Start( F:in Positive )  is --Rendezvous
         begin
               aux2:=f;
              answer2:=aux2+10;
    end start;
    procedure finish(result:out positive) is
         begin
             Result := Answer2;
    end Finish;
  end proceso_datos;

begin
    Thread_1.Start(5);
     Put("El primer resultado es: ");
    Thread_1.Finish( trabajo ); --Se imprime el resultado
    Put( trabajo ); New_Line;
    Thread_2.Start(8);
    Put("El segundo resultado es: ");
```

```
    Thread_2.Finish( trabajo2 ); --Se imprime el resultado
    Put( trabajo2 ); New_Line;
    null;
    new_line;
end protegido;
```

ANEXO D(Ejemplo de Citas en Ada)

Este programa en su interior realiza lo mismo que el programa de objetos protegidos, la diferencia radica en que las tareas son declaradas de manera convencional con la diferencia que tanto en su declaración como en su cuerpo se les especifica que deben recibir un dato por medio de una cita.

```
with ada.text_io,Ada.Integer_Text_Io;
use ada.text_io,Ada.Integer_Text_Io;
--with Ada.calendar;
---use Ada.calendar;

procedure citas is

task type lectura_datos is       --Declaración de la tarea
   entry Start( F:in Positive ); --Rendezvous(cita)
   entry Finish( Result:out Positive ); --Rendezvous(cita)
end lectura_datos;

task type proceso_datos is
   entry Start( F:in Positive );
   entry Finish( Result:out Positive );
end proceso_datos;
----------------------------- Declaración Variables Globales-------------
Thread_1 : lectura_datos;--Declaración de la variable Thread_1
Thread_2 : proceso_datos; --Declaración de la variable Thread_2
trabajo  : positive;--Declaración de trabajo y trabajo2 de tipo positivo
trabajo2 : positive;

------------------------  Cuerpo de la Tarea Lectura Datos -------------
task body lectura_datos is
    aux     : positive; --Declaración de aux y answer como positivas
    Answer  : Positive;
begin
   accept Start( F:in Positive ) do--Accept recibe el dato que se le ha
```

```
        aux   := F;               --que se le ha mandado a la tarea
   end Start;
    answer:=aux*2;
    accept Finish( Result:out Positive ) do--Retorna el resultado de la
tarea
          Result := Answer;
   end Finish;
 end lectura_datos;
--------- Cuerpo de la Tarea Proceso Datos ---------------------------

task body proceso_datos is
aux2     : positive;--Declaración de las variables locales aux2,answer2
Answer2  : Positive;
   begin
       accept Start( F:in Positive ) do--acepta el dato a procesar en la
         aux2  := F;                   -- tarea
      end Start;
   answer2:=aux2 + 10;                 -- procesamiento del dato
   accept Finish( Result:out Positive ) do --Retorno del dato procesado
          Result := Answer2;
   end Finish;
end proceso_datos;
------------------- Programa Principal---------------------------------
begin
   Thread_1.Start(5);--Se envía el dato a la cita por medio del hilo
    Put("El primer resultado es: ");
    Thread_1.Finish( trabajo ); --Se obtiene el dato procesado y se
imprime
    Put( trabajo ); New_Line;
    Thread_2.Start(8);
    Put("El segundo resultado es: ");
    Thread_2.Finish( trabajo2 );
    Put( trabajo2 ); New_Line;
    null;
    new_line;
end citas;
```

ANEXO E(Programa en Ada de monitoreo de tanques en Tiempo real)

```
with ada.text_io;
use ada.text_io;
with System.Machine_Code;
use System.Machine_Code;
with Interfaces;
with Ada.Real_Time;
use Ada.Real_Time;

procedure PC is
dato: integer;
datosp: integer;
aux:string(1..3);
 x:character;
 start:integer:=0;
----------Procedimiento de escritura en los puertos------------------------------
procedure  escritura(dir:in interfaces.unsigned_16;dato: in interfaces.Unsigned_8) is
   begin
      ASM ("OUTB %%DX",
      No_Output_Operands,
     (interfaces.Unsigned_8'Asm_Input  ("a", dato),
      interfaces.Unsigned_16'Asm_Input ("d", dir)));
end escritura;
 ----------procedimiento de lectura en los puertos------------------------------
function leer(dir: in interfaces.unsigned_16) return interfaces.Unsigned_8  is
   dato: interfaces.Unsigned_8;
   begin
      ASM ("INB %%DX",
      INTERFACES.Unsigned_8'Asm_Output ("=a",dato),
```

```
      INTERFACES.Unsigned_16'Asm_Input ("d", dir),
      Volatile => True);
   return dato;
end leer;

--************************ enviar *********************************

  procedure send(dato: in interfaces.unsigned_8) is
    x: integer;
    vector: array (1..8) of integer;
    begin
      escritura(771,0);   --arranca con SCK=0 y SO=0
      x:= integer(dato);

       for i in 1..8 loop   -- para enviar 8 bits
         if x mod 2 = 1 then    -- Si el modulo 2 del dato es 1 envía uno
           vector(i):=1;
         else
           vector(i):=0;
         end if;
         x:=x/2; --divide el dato entre dos (aspectos de la conversión a binario).
       end loop;

       for i in 1..8 loop   -- para enviar 8 bits
         if vector(9-i) = 1 then
           escritura(771,2);      --SCK=0 y SO=1
           escritura(771,6);      --SCK=1 y SO=1
           escritura(771,2);      --SCK=0 Y SO=1
         else
           escritura(771,0);      --SCK=0 y SO=0
```

```
            escritura(771,4);      --SCK=1 y SO=0
            escritura(771,0);      --SCK=0 y SO=0
          end if;
        end loop;
       escritura(771,0);  --escribe cero en pto SCK=0 , SO=0
  end send;

--******************** leer spi************************************

   function leer_spi(dir: in integer) return integer is
     pto: integer;
     dato1: integer;
     begin
       dato1:=0;
       escritura(771,0);   --inicializa en cero puerto ISA.
       send(3);         --envía comando serial a la spi de lectura.
       send(interfaces.Unsigned_8(dir));         --envía la dirección del registro.
       for i in 0..7 loop
         escritura(771,4);   --SCK=1, SO=0
         pto:=integer(leer(770));
         dato1:=dato1+(pto*2**(7-i));
         escritura(771,0);  --SCK=0, SO=0
       end loop;
      return dato1;
   end leer_spi;
--******************************** escribir spi ****************************
   procedure escribir_spi(dire: in interfaces.unsigned_8;dato : in interfaces.unsigned_8 ) is
     begin
       escritura(771,0);
       send(2);
```

```
        send(dire);
        send(dato);
        escritura(771,1);
   end escribir_spi;

--************************* bit spi ************************************
procedure bit_spi(dato: in interfaces.unsigned_8;masc : in interfaces.unsigned_8; dire:
in interfaces.unsigned_8) is
   begin
      escritura(771,0);
      send(5);
      send(dire);
      send(masc);
      send(dato);
      escritura(771,1);
    end bit_spi;

--******************** reset spi ************************************

   procedure reset_spi(dato: in interfaces.unsigned_8) is
    begin
      escritura(771,0);
      send(dato);
      escritura(771,1);
     end reset_spi;

--********************** rts spi ***************************************

   procedure rts_spi(dat: in interfaces.unsigned_8) is
      begin
```

```
        escritura(771,0);
        send(dat);
       escritura(771,1);
   end rts_spi;


--**************************proceso de inicialización*********************
procedure inicializacion is

begin
    bit_spi(128,224,15);      -- modo de configuración
     reset_spi(192);     --REINICIA EL MCP2510
     escribir_spi(48,03);   -- mensaje de alta prioridad, en txb0
     escribir_spi(96,32);   --RXBOCTRL recibe solo tramas estándar
     escribir_spi(112,32);   -- RXB1CTRL recibiendo solo tramas estándar
     escribir_spi(132,15);
     escribir_spi(53,1);      --dlc3 solo transmite 1 byte
     escribir_spi(42,1);   --cnf1
     escribir_spi(41,160); --cnf2
     escribir_spi(40,2);   --cnf3    configuracion de velocidad
     escribir_spi(49,0);   --id estandar parte alta
     escribir_spi(50,128);  --id estandar parte baja(bits 2,1,0); y bits 17
     escribir_spi(65,0);   --id estandar parte alta
     escribir_spi(66,160);  --id estandar parte baja(bits 2,1,0); y bits 17
     escribir_spi(48,3);   --control del bufer de transmision cero
     escribir_spi(64,3);   --control del bufer de transmision uno.
     escribir_spi(96,96);  --control del bufer de recepcion cero.
     escribir_spi(112,96); --control del bufer de recepcion uno
     escribir_spi(15,4);   --registro de control. tabajo en modo normal
     escribir_spi(24,0);   --parte alta del filtro 5
```

```
      escribir_spi(25,32);  --parte baja del filtro 5
      escribir_spi(20,0);   --parte alta del filtro 4
      escribir_spi(21,64); recibiendo identificador 2 = nivel
      escribir_spi(16,0);   --parte alta del filtro 3
      escribir_spi(17,96);  --parte baja del filtro 3
      escribir_spi(8,0);   --parte alta del filtro 2
      escribir_spi(9,192);  --parte baja del filtro 2
      escribir_spi(4,0);   --parte alta del filtro 1
      escribir_spi(5,224); --parte baja del filtro 1
      escribir_spi(43,3);  --parte baja del filtro 4,recibiendo identificador 2 = nivel
      bit_spi(0,224,15);   -- modo normal
end inicializacion;
--*****************************Tarea *********************************
task Procesamiento;
task body recepcion_datos is
   periodo: time_span := milliseconds(125);
   siguiente : time;
   begin
     inicio:=clock;
     siguiente:=clock;
   loop
         inicializacion;
         dato:=leer_spi(98);
         if dato = 192 then
            scribir_spi(70,interfaces.unsigned_8(dato));   -- byte de datos
             rts_spi(130);
         end if;
         if dato = 32 then
               dato:=leer_spi(102)*2;
               new_line;
```

```
                put("nivel en milimetros:") ;
                put(dato'img);
                new_line;
            end if;
            bit_spi(0,1,44); -- limpia la bandera de recepción para poder recibir un nuevo dato.
           siguiente := siguiente + periodo;
          delay until siguiente;
      end loop;
 end procesamiento;
--*****************************************************************************
  begin
  loop
  New_line;
  Put_Line
("*****************************************************************************");
   Put ("  ");
   New_Line;
   Put_Line ("SISTEMA DE MEDICION DE NIVEL DE UN TANQUE EN TIEMPO
REAL");
   New_Line;
   New_Line;
   New_Line;
   New_Line;
   Put_Line
("*****************************************************************************");
   New_Line;
   New_Line;
    end loop;
  null;
end PC;
```

ANEXO F (Instalación del RTLGnat 1.0)

1.Para la instalación de RTLGnat se requiere de Linux Redhat 7.2 o superior, Rtlinux 3.2 y Gnat 3.15, además el núcleo del Linux 2.4.18 y el parche para memoria dinámica bigphysarea.

2. Bajar los archivos apropiados:

- Núcleo de linux 2.4.18. **http://www.kernel.org/**
- RTLGnat-1.0. **http://bernia.upv.es/rtportal/apps/rtl-gnat/RTLGnat-1.0/**
- Rtlinux-3.2. **ftp://ftp.rtlinux-gpl.org/**
- Gnat-3.15. **ftp://cs.nyu.edu/pub/gnat/**
- Bigphysarea. **http://www.polyware.nl/~middelink/En/hob-v4l.html**

3. Desempaquetar los archivos:

 # rm –rf /usr/src/rtlinux

 # mkdir /usr/src/rtlinux

 # cd /usr/src/rtlinux

 # tar –xzf /var/tmp/linux-2.4.18.tar.gz

 # tar –xzf /var/tmp/rtlinux-3.2-pre2.tar.gz

 # tar –xzf /var/tmp/bigphysarea-2.4.4.tar.gz

 # cd /usr/src

 # tar –xzf /var/tmp/gnat-3.15p-i686-pc-redhat71-gnu-bin.tar.gz

4. Aplicar el parche para memoria dinámica:

 # cd /usr/src/rtlinux/linux

 # patch –p1 < /usr/src/rtlinux/bigphysarea-2.4.4/bigphysarea.diff

5. Aplicar el parche de rtlinux:

patch –p1 < /usr/src/rtlinux/rtlinux-3.2-pre2/patches/kernel_patch-2.4.18-rtl-3.2-pre2

6.Configure el kernel de linux:

 # make menuconfig

Code maturity level options ---> yes

Processor type and features ---> Seleccione el procesador correspondiente a su máquina o en su defecto 386

[*] Symmetric multi-processing support

Loadable module support --->

[*] Enable loadable module support

[ ] Set version information on all symbols for modules

[ ] Kernel module loader

General setup ---> accept defaults

Plug and Play support ---> accept defaults

Plug and Play support ---> accept defaults

Networking options ---> accept defaults

SCSI support ---> accept defaults

Network device support --->

Ethernet (10 or 100Mbit) --->

[*] 3COM cards

<M> 3c590/3c900 series (592/595/597) "Vortex/Boomerang" support

Kernel hacking --->

[*] Magic SysRq key

7. Compile e instale los modules del kernel o núcleo y los módulos:

 # make dep

 # make bzImage

 # make modules

 # make modules_install

 # cp arch/i386/boot/bzImage /boot/rtzImage

8. Desempaque RTLGnat 1.0:

 # cd /usr/src

 # tar xfj /var/tmp/RTLGnat-1.0

9. Aplique el parche del RTLGnat 1.0

 # cd /usr/src/rtlinux/rtlinux-3.2-pre2

 # patch –p1 < /usr/src/RTLGnat-1.0/patch/rtlgnat-1.0-RTLINUX-3.2-pre2.patch

10. Configure y compile rtlinux, dejando las opciones que aparecen activas por defecto y activando:

 # cd /usr/src/rtlinux/rtlinux-3.1

 # ln –sf /usr/src/rtlinux/linux linux

 # make menuconfig

Deje las opciones que aparecen activas por defecto y active:

- Priority inheritance (POSIX Priority Protection)
- Dynamic memory manager
- POSIX Signals support
- Floating Point support

luego:

 # make dep

 # make

 # make devices

 # make install

11. Configure el lilo:

Vaya a etc/lilo.conf y adicione lo siguiente:

```
image=/boot/rtzImage
```

```
label=rtlinux
read-only
root=/dev/had1
append="bigphysarea=1024"
```

12. Instale el lilo y reinicie el computador

 # /sbin/lilo

 # /sbin/shutdown –r now

13.Instale el gnat 3.15

cd /usr/src/gnat-3.15p-i686-pc-redhat71-gnu-bin

 # ./doconfig

Luego seleccione la opción 2

y después

 # ./doinstall

14.Vaya a /usr/src/RTLGnat-1.0 y edite el archivo Makefile, colocando el path tanto de linux como de rtlinux.

15. La versión de gnat debe estar al comienzo del path

 # export PATH=/usr/gnat/bin:$PATH

16. Compile el RTLGnat:

cd /usr/src/RTLGnat-1.0

 # make

17. Compile los ejemplos de RTLGnat-1.0

 # cd /usr/gnat/rtl_examples

 # make

18. Antes de correr las aplicaciones de ada sobre rtlinux, se deben cargar los modulos de Tiempo Real:

 # rtlinux start

19.Para compilar y correr nuestras propias aplicaciones en rtlinux:

 # rtlgnatmake my_app.adb

Una vez creados nuestros objetos, se deben montar los modulos de Tiempo Real

 # rtlinux start

luego monte su aplicación sobre rtlinux:

 # rtload my_app

Para que la aplicación terminé su ejecución:

 # rtunload my_app

ANEXO G (Plano tarjeta ISA para la comunicación con la red CAN)